IMPLEMENTACIÓN DE UN PROTOTIPO DE COMUNICACIONES REMOTO QUE PERMITA GEO-REFERENCIAR DISPOSITIVOS CON TECNOLOGÍA GPS EN LAS UNIDADES TECNOLOGICAS DE SANTANDER

JORGE A. ANTOLINES ESTUPIÑAN

YEIDSON MANTILLA GELVEZ

ASESOR

MG. FRANCISCO JAVIER DIETES CARDENAS

UNIDADES TECNOLOGICAS DE SANTANDER

FACULTAD CIENCIAS NATURALES E INGENIERÍAS

INGENIERÍA EN TELECOMUNICACIONES

BUCARAMANGA

2013

RESUMEN

El presente proyecto se basa en la implementación de un prototipo de comunicaciones remoto, que permita Geo-referenciar dispositivos con tecnología GPS, que funciona con un software y un hardware libre llamado Arduino Uno. De esta manera el GPS permite captar las señales emitidas por el satélite dando la latitud, longitud, velocidad, altura y tiempo de los vehículos que tengan el dispositivo, y esta información a su vez es almacenada en una memoria MicroSD, que se puede visualizar a través de la plataforma llamada Google Earth.

TABLA DE CONTENIDO

CAPÍTULO 3: DISEÑO E IMPLEMENTACIÓN

CAPÍTULO 4: CONCLUSIONES

CAPÍTULO 5: REFERENCIAS Y ANEXOS

TABLA DE FIGURAS

GLOSARIO

BOOTLOADER: es el gestor de arranque que permite sketch a la placa de arduino.

BUG: errores introducidos al teclear el código.

BYTES: es una secuencia de bits contiguos, cuyo tamaño depende del código de información o código de caracteres en que sea definido

DRIVER, DRIVERS: software de soporte. Es usado para la conexión entre las placas y el sistema operativo del PC.

DSP: procesador digital de señal, es un sistema basado en un procesador o microprocesador que posee un conjunto de instrucciones, un hardware y un software optiminizados para aplicaciones que requieran operaciones numéricas a muy alta velocidad.

GPS: Sistema de posicionamiento global.

GSM: Sistema Global para comunicaciones móviles.

HDL: Hardware description language, es un lenguaje definido por el IEEE (Institute of Electrical and Electronics Engineers), usado por ingenieros para describir circuitos digitales.

LED: Diodo emisor de luz, transforma energía eléctrica en luz.

MICROSD: corresponden a un formato de tarjeta de memora flash, más pequeña que la miniSD, desarrollada por SanDisk.

PCI: interconexión de componentes periféricos en un bus de ordenador estándar para conectar dispositivos periféricos directamente a su placa base.

PPS: terminal de GPS modelo EM-406, este proporciona un pulso por segundo de salida de la placa Arduno que está sincronizada con el tiempo del GPS.

PWM: pulso de amplitud modulada, es como el arduino simula una salida analógica.

RX: en la placa GPS EM406, es el principal canal para recibir comandos del software.

SHIELD: tarjeta para las placas que se montan sobre arduino.

SKETCH: uno de los programas que se ejecutan dentro del arduino.

SPI: Internet periférico serie, protocolo que se utiliza en casos de comunicación rápida entre la placa Arduino y varios periféricos.

SMS: Servicio de mensajes cortos.

TX: en el GPS EM406, es el canal de transmisión principal para la salida de datos de navegación y medición para el usuario.

UART: transmisor-receptor asíncrono Universal controla los puertos y dispositivos serie. Se encuentra integrado en la placa base o en la tarjeta que se adapta al dispositivo.

USB: Bus Universal en serie es un estándar industrial desarrollo en la década de los 90's que define los cables, conectores y protocolos usados en un bus para conectar, comunicar y proveer de alimentación eléctrica entre ordenadores, periféricos y dispositivos electrónicos.

WAYPOINT: Posición de un único lugar sobre la superficie de la tierra expresada por sus coordenadas.

CAPÍTULO 1

PRESENTACIÓN DEL PROYECTO

1.1 INTRODUCIÓN

En el transcurso de la carrera de Ingeniería en Telecomunicaciones, se estudiaron temas que involucran tecnologías de comunicaciones inalámbricas como wifi, radio-enlaces, GSM y comunicaciones avanzadas como enlaces satelitales, sistemas GPS y programación de dispositivos móviles como programación en Netbeans, la unión de estos conocimientos permite crear un prototipo flexible y escalable en este caso para geo-referenciar cualquier dispositivo que teng}a un sistema de posicionamiento global.

El objeto de este proyecto es el diseño y construcción de un prototipo que transmita en tiempo real la geo-posición de cualquier dispositivo utilizando el módulo GPS.

Para alcanzar el desarrollo de este prototipo se trabaja con base al open-source o código abierto tanto para los dispositivos electrónicos (módulos y tarjetas) como para el código de programación de los dispositivos programables (microcontroladores), dispositivos encargados de controlar el funcionamiento del módulo.

Los módulos que se trabajan son de la plataforma Arduino.

El desarrollo del trabajo arroja resultados que pueden ser el soporte para el desarrollo de futuras aplicaciones creadas bajo open-source en el programa de Ingeniería en Telecomunicaciones.

1.2 DESCRIPCIÓN DEL PROYECTO

Debido a la creciente variedad de dispositivos inalámbricos y nuevos protocolos surgen soluciones flexibles y escalables de código abierto, que se adaptan a las necesidades de los diseñadores y programadores interesados en esta área del conocimiento tecnológico.

La comunidad científica ha validado y certificado la eficiencia de estos dispositivos inalámbricos en este caso Arduino, que es una plataforma de hardware de código abierto, basada en una sencilla placa con entradas y salidas analógicas y digitales, en un entorno de desarrollo que está basado en el lenguaje de programación Processing.

Es un dispositivo que conecta el mundo físico con el mundo virtual, o el mundo analógico con el digital. Sus creadores son el zaragozano David Cuartielles, ingeniero electrónico y docente de la Universidad de Mälmo, Suecia y Massimo Banzi, italiano, diseñador y desarrollador Web. El proyecto fue concebido en Italia en el año 2005.[1]

Actualmente esta plataforma permite soluciones integradas para la transmisión de datos en redes LAN, WIFI, BLUETOOTH y redes de telefonía móvil (GSM), entre otros. Para este proyecto se tiene en consideración el módulo Arduino como solución a la transmisión y recepción de datos provenientes de un dispositivo GPS.

El objetivo primordial de este proyecto es integrar la solución mencionada anteriormente a un sistema que contenga una base de datos que permita visualizar la información geográfica en tiempo real.

Este proyecto es el inicio de una línea de investigación en el área de dispositivos lógicos programables que integre conocimientos de áreas como comunicaciones móviles, programación de dispositivos móviles y comunicaciones avanzadas. El prototipo final permite la flexibilidad en la programación y conectividad con diversos estándares inalámbricos.

La metodología se basa en los lineamientos del método científico haciendo énfasis en pruebas y ajustes del prototipo final en el laboratorio.

---

[1] David Cuartielles "¿Qué es Arduino?", Internet (http://proyectoarduino.wordpress.com/%C2%BFque-es-arduino/)

1.3 OBJETIVOS

1.3.1 OBJETIVO GENERAL

- Implementar un prototipo de comunicaciones remoto que permita geo-referenciar dispositivos con tecnología GPS.

1.3.2 OBJETIVOS ESPECÍFICOS

- Identificar las propiedades del estándar de la tarjeta Arduino como interfaz de comunicaciones para el módulo GPS.
- Diseñar un prototipo utilizando el módulo Arduino y receptor GPS, que permita la recepción en tiempo real de sus coordenadas geográficas (Latitud y Longitud).
- Diseñar la aplicación para celulares con tecnología superior o igual a 3G, que permita geo-referenciar las coordenadas provenientes del módulo.
- Crear un manual de usuario dentro de la herramienta de monitoreo.

1.4 ALCANCES Y LIMITACIONES

El proyecto está enfocado en el desarrollo de un prototipo en el que se puede geo-referenciar un nodo, integrando software, hardware a la plataforma abierta Arduino para la captura y envío de datos.

El proyecto se lleva a cabo en la planta física de las Unidades Tecnológicas de Santander, donde se llevan a cabo las pruebas del prototipo.

El prototipo no garantiza una adecuada cobertura en interiores debido a la tecnología GPS.

Los módulos que se utilizan están sujetos a la disponibilidad en el mercado y compatibilidad con los dispositivos locales.

CAPÍTULO 2

MARCO TEÓRICO

Antes de comenzar a realizar la descripción detallada del proyecto, se comienza con el significado de cada uno de los componentes del proyecto, continuando con Arduino Uno, sus partes, bloques y microcontrolador, luego la definición de una shieldSD y su forma de conexión, al igual que una shieldGPS, finalizando con la obtención de los resultados, gracias a la unión de estos tres componentes, complementados por el sketch.

2.1 GPS:

El Sistema de Posicionamiento Global (GPS), permite determinar la posición de un objeto o persona con coordenadas de latitud, longitud y altura.

Es decir que para conocer la ubicación de un objeto o persona es necesario de un receptor GPS, que mide la distancia de cada satélite a la antena del receptor. De esta manera, para reconocer la distancia los satélites envían ondas de radio a 300.000 kilómetros por segundo y de igual forma, mide el tiempo entre el momento que sale la señal y el momento en que llega al receptor.

El GPS se creó en el departamento de defensa de Estados Unidos a finales de la "Guerra Fría" con objetivos militares y su uso pasó a náutica y aviación.

Inicialmente la cobertura del GPS no era completa, ya que faltaba ubicar en órbita varios satélites. Además el costo era demasiado alto, lo que impedía que todo el público objetivo tuviese acceso a él.

En la actualidad el Sistema de Posicionamiento Global funciona de manera completa, operativa y es bastante asequible para el mercado que va dirigido.

El funcionamiento del GPS se basa entonces, en el proceso y recepción de los datos que se emiten a través de NAVSTAR, que es una serie de 24 satélites, que orbita a una altura de 20.200 kilómetros aproximadamente por encima de la superficie terrestre y a su vez cada uno de los satélites da vueltas al planeta cada 12 horas.

En este sentido, cabe resaltar que para que un receptor GPS, registre datos a cualquier hora del día o de la noche y en cualquier lugar (con condiciones metereológicas), debe haber en todo momento cinco satélites a la vista en cualquier zona, pues de esta manera se facilita su ubicación y registra las señales emitidas con un mínimo de tres satélites.

Con un satélite se tiene una distancia, esto quiere decir que la posición a determinar puede estar en cualquier punto de una esfera hueca a una distancia X desde el satélite. Ver figura 2.1.

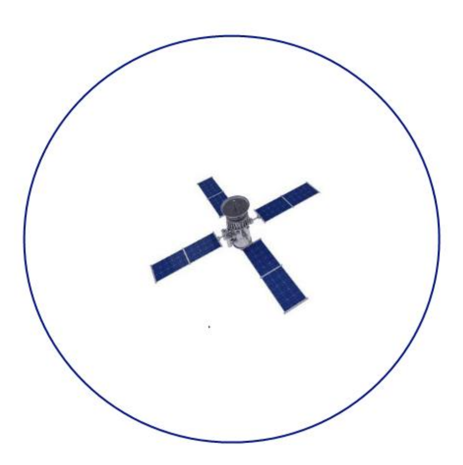


Figura 2.1 Área cobertura de un satélite.

Tecnoproject. Seguridad Electrónica e informática. [Fotografía]. (2010). Tomado de http://biblioinstruccion.blogspot.com/2010/12/como-citar-imagenes-segun-el-estilo-apa.html

Con dos satélites el punto puede estar en algún lugar del círculo de intercepción de las dos distancias de los satélites. Ver figura 2.2.

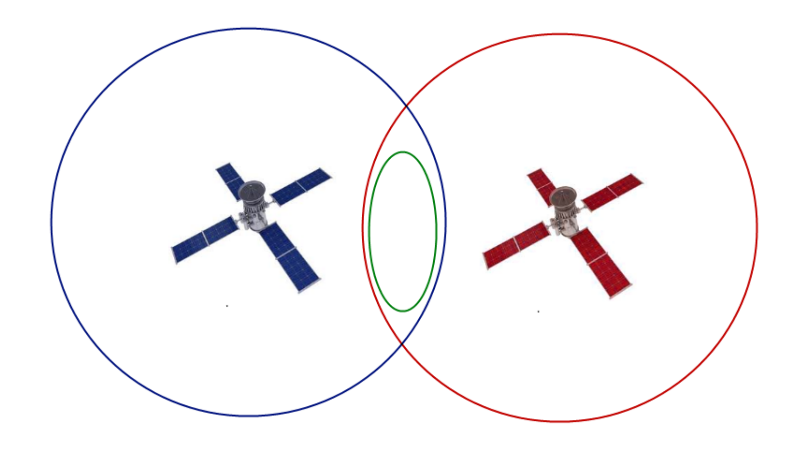


Figura 2.2 Área cobertura dos satélites

Tecnoproject. Seguridad Electrónica e informática. [Fotografía]. (2010). Tomado de http://biblioinstruccion.blogspot.com/2010/12/como-citar-imagenes-segun-el-estilo-apa.html

Midiendo la distancia desde tres satélites se puede reducir a dos puntos en el espacio el lugar en que se encuentre; lo que quiere decir que una posición es verdadera y la otra es falsa. El sistema en sí puede determinar cuál punto es incorrecto porque no está cerca de la Tierra. Ver figura 2.3.

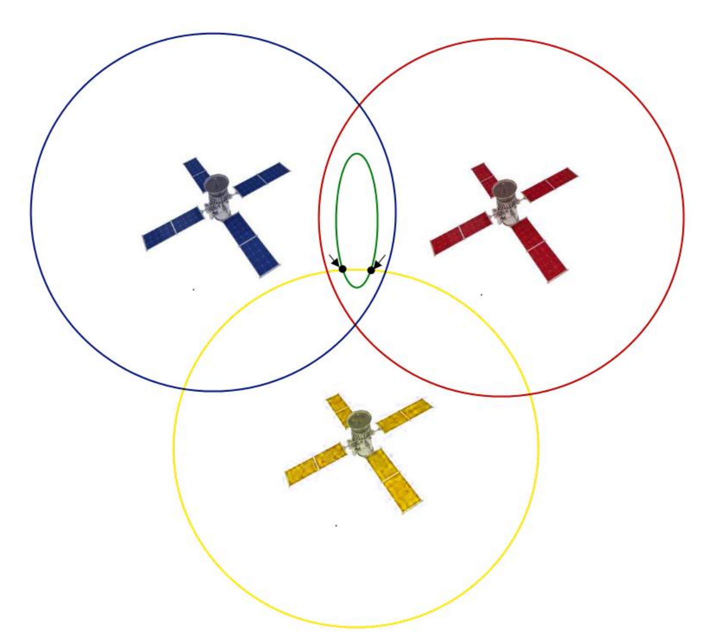


Figura 2.3. Área de cobertura de tres satélites

Tecnoproyect. Seguridad Electrónica e informática. [Fotografía]. (2010). Tomado de
http://biblioinstruccion.blogspot.com/2010/12/como-citar-imagenes-segun-el-estilo-apa.html

Finalmente midiendo la distancia a cuatro satélites se puede determinar la posición de un punto. Ver figura 2.4.

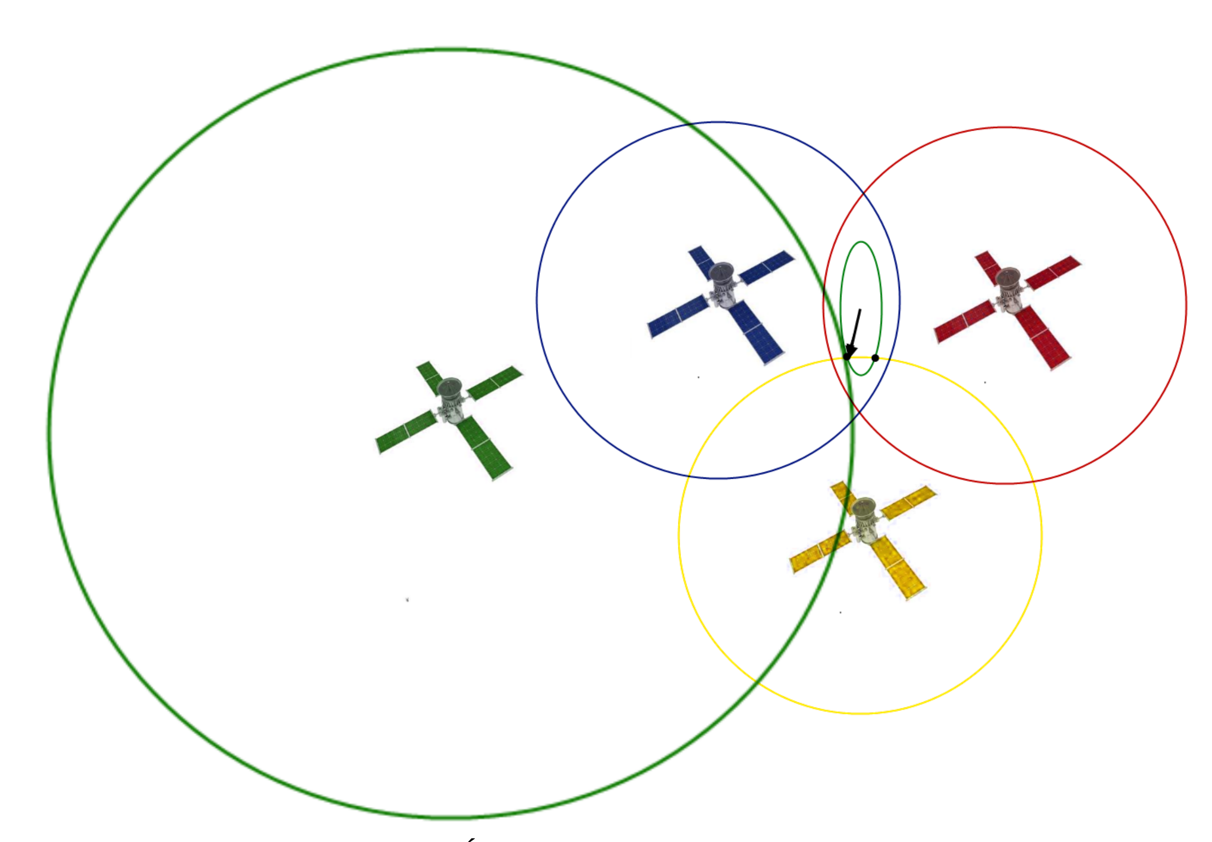


Figura 2.4. Área cobertura de cuatro satélites

Tecnoproyect. Seguridad Electrónica e informática. [Fotografía]. (2010). Tomado de
http://biblioinstruccion.blogspot.com/2010/12/como-citar-imagenes-segun-el-estilo-apa.html

Los receptores GPS están diseñados de acuerdo al tipo de aplicaciones como: Geodesia, Topografía, Navegación Marítima, Navegación Aérea, Navegación Terrestre, Cartografía, AVL (Localización Automática de Vehículo). Cada receptor posee funciones y precisiones específicas de acuerdo a su aplicación.

La precisión de los GPS varía desde 100 metros a precisión milimétrica esto depende de la cantidad de satélites que se utilicen para dicha medición.

2.1.1 FUNCIONAMIENTO DEL GPS

Los satélites de la constelación GPS constantemente emiten dos códigos de datos diferentes en formato digital, que a su vez son transmitidos por señales de radio.

El primero de los códigos es para uso exclusivamente militar y no los captan los receptores GPS civiles. El segundo código es de uso civil y transmite dos series de información, que son denominadas Almanaque y Evento, y se encargan de informar el estado operativo de funcionamiento del satélite, es decir, su situación orbital, la fecha y la hora.

De esta manera, se debe tener en cuenta que los satélites emiten sus propios Almanaques y Eventos y tienen un código de identificación propio para cada satélite. Estos últimos poseen unos relojes atómicos que determinan una precisión casi total, con un margen de error de un segundo cada 70.000 años.

El GPS debe tener en su memoria del Almanaque y los Eventos actualizadas, en el momento en que no lo estén automáticamente se actualizan, cuando el receptor sintonice las señales emitidas por un mínimo de tres satélites, y de esta manera sabrá dónde ubicar los satélites en el firmamento.

Los datos obtenidos de los satélites para geo-referenciar cualquier objeto, son latitud y longitud, estas coordenadas son expresadas en minutos y/o segundos, en una sentencia NMEA

$GPRMC,054235.000,A,4522.0289,N,00724.5210,E,0.39,65.46, 070613,,,A*44

- **044235.000** hora GMT (05:42:35)
- **A** dato fijado y correcto
- **4522.0289** long (45º 22.0289´)
- **N** Norte

- **00724.5210** es la lat (7º 24.5210´)
- **E** Oeste
- **0.39** velocidad (nudos)
- **65.46** grados
- **070613** fecha (7 de junio del 2013)

3.6.2 NOMBRES Y DESCRIPCIÓN DE LAS FUNCIONES DE UN GPS

Posición:

Facilita la ubicación casi exacta del GPS, que en este caso éste debe captar señales emitidas al menos por tres satélites.

Altura:

Cuando el GPS capta cuatro o más satélites, indica la altura sobre el nivel del mar.

Tiempo:

Al iniciar el GPS, indica la hora y fecha aunque aún no haya recibido señales satelitales, sin embargo, si éstas se dan el GPS registra la hora exacta.

Punto de Referencia:

El punto de referencia es la demarcación que se sitúa gráficamente en un plano o mapa, en este para los GPS, puede ser un punto de inicio, de destino o de ubicación de una ruta u objeto.
El punto de referencia se puede realizar y guardar en una memoria, éste puede modificarse o borrarse y también se puede hacer una sucesión, que permita determinar un recorrido; esto se denomina Tracker de Puntos o de Ruta.

Distancia:

Para calcular la distancia entre dos puntos, es necesaria la utilización de mínimo de dos satélites y un tercero de confirmación.

En la siguiente imagen se observa detalladamente los datos anteriormente explicados.



Figura 2.5. Captura de datos
ANTOLINES, Jorge. Prueba. [Fotografía]. (2013).

2.2 MICROCONTROLADORES

Un microcontrolador es *"un circuito integrado programable que contiene todos los componentes de un computador (unidad central de procesamientos, memoria y unidades de Entrada y Salida) y se emplea para realizar una tarea determinada para la cual ha sido programado"*[2].

De la basta familia de microcontroladores que existen, en este caso el ATMEGA328 utilizado por Arduino, la mayor ventaja de éste es que no necesita ningún tipo de quemador, ya que Arduino lo incorpora sin problema alguno.
El concepto del microcontrolador es el mismo que cualquier PIC, que requiere un script que será almacenado en su memoria EEPROM, cuya función será la requerida por el diseñador o programador. Se anexa datasheet.

[2] Usuarios.multimania.es/sparta/studies0.html

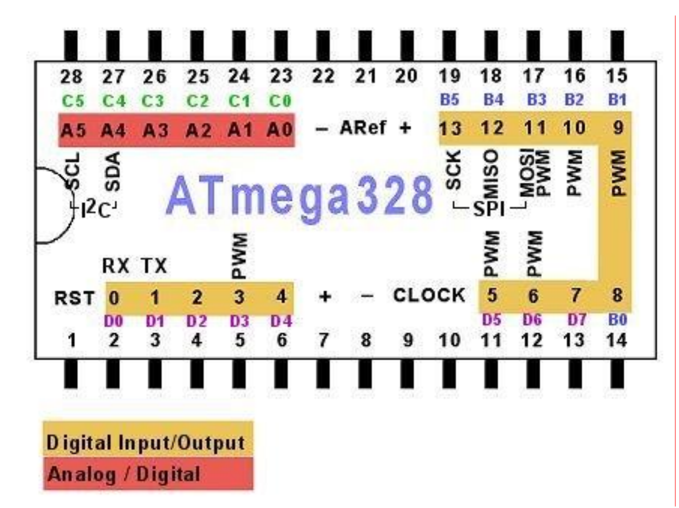


Figura 2.6. Microcontrolador ATMEGA 328

Hobby Electronics. " Arduino Atmega328 Pinout" [Fotografía]. Tomado de: http://www.hobbytronics.co.uk/arduino-atmega328-pinout

Flash (Kbytes):	32 Kbytes
Número de pines:	32
Max. Frecuencia de operación:	20 MHz
CPU:	AVR de 8 bits
N º de Canales Touch:	16
Hardware Qtouch adquisición:	No
Max I / O Pins:	23
Ext. Interrupciones:	24
Speed USB:	No
Interfaz USB:	No

Tabla 2.1. Parámetros principales ATMEGA 328

2.3 PUERTO DE COMUNICACIONES USB

Traduce Universal Serial Bus (Bus Universal Serie), que es un puerto que sirve para conectar periféricos a un ordenador.

"Los microcontroladores son los que han permitido la existencia de este sistema de comunicación. Es un sistema que trabaja por polling (monitorización) de un conjunto de periféricos inteligentes por parte de un amo, que es normalmente un

computador personal. Cada modo inteligente está gobernado inevitablemente por un microcontrolador”. [3]

USB fue creado con el fin de estandarizar la conexión de periféricos como los teclados, joysticks, mouse, escáneres, cámaras digitales, teléfonos móviles, impresoras, etc, es decir, que fue creado con el fin de suprimir la necesidad de tener tarjetas separadas para poner en los puertos Bus ISA o PCI y mejorar las capacidades Plug and Play, logrando que estos dispositivos sean conectados o desconectados al sistema sin tener que reiniciarlo.

2.4 DEFINICIÓN DE SOFTWARE LIBRE

El Software libre se define como el software que respeta la libertad de los usuarios y la comunidad, pues éstos tiene la posibilidad o libertad de copiar, distribuir, estudiar, modificar y mejorar el software. Los usuarios de esta manera pueden controlar el programa y lo que éste hace.

Los usuarios de un software libre tienen cuatro libertades esenciales:

- Libertad de ejecutar el programa para cualquier propósito (libertad 0).
- Libertad de estudiar cómo funciona el programa, modificar o cambiar algo, con el fin de que el programa haga lo que el usuario quiera (libertad 1). El acceso al código fuente es necesario.
- Libertad de redistribuir copias para ayudar a los demás (libertad 2).
- Libertad de distribuir copias de sus versiones modificadas a otras personas (libertad 3). Permite entonces que todas las personas se beneficien de los cambios y también en este caso el acceso al código fuente es una condición necesaria.

[3] WARREN, Jhon David. “Arduino Robotics”:
http://books.google.com.co/books?id=tBdKnLcf2oEC&printsec=frontcover&dq=ARDUINO&hl=es-419&sa=X&ei=G0cNUtf7Dqew2AW024DgBQ&ved=0CFMQ6AEwBQ#v=onepage&q=ARDUINO&f=false

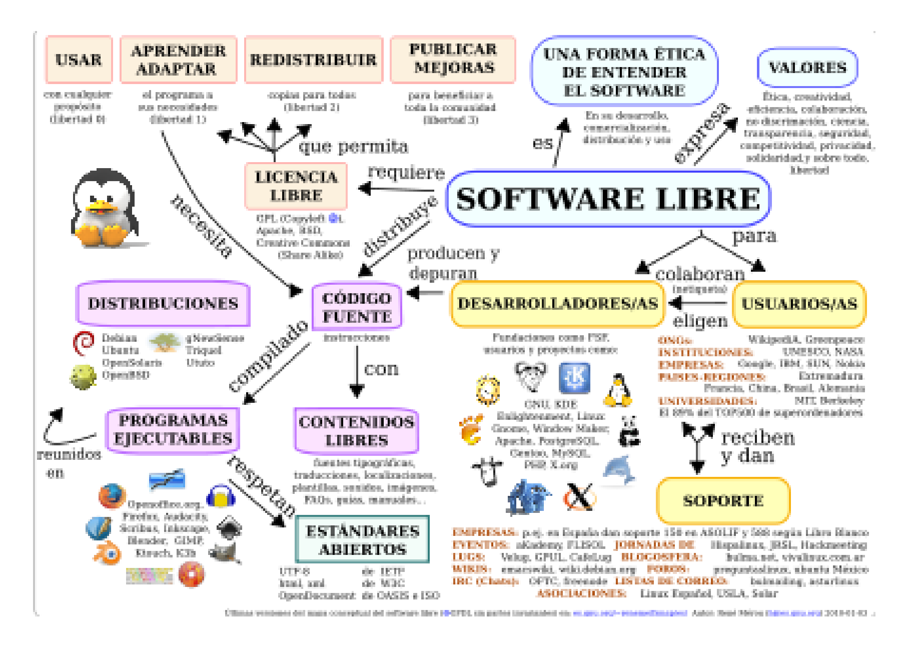


Figura 2.7. Definición de Software libre

CENA, Alberto. "Definición de Software libre". [Fotografía]. (2011). Tomado de: http://blogfolio-de-albertocena.blogspot.com/2011/06/definicion-de-software-libre.html

2.5 HARDWARE LIBRE

Los dispositivos para la interconexión, programación y obtención de información utilizados son los siguientes:

1. Arduino UNO.
2. GPS EM406
3. Módulo SD
4. Cable USB
5. Computador

2.6 ARDUINO

"Arduino es una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware flexibles y fáciles de usar"[4].

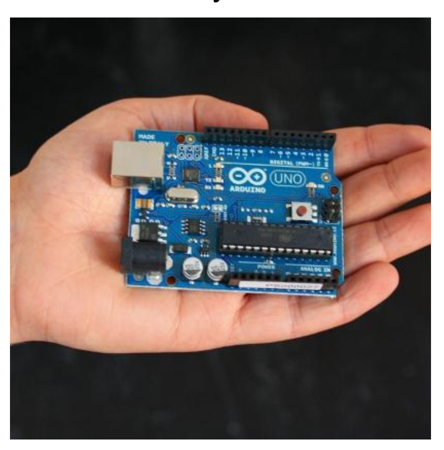


Figura 2.8 Placa base de Arduino

Seguridad Doméstica. "Domótica con Arduino". [Fotografía] (2012). Tomado de: http://www.livemodern.org/domotica-con-arduino-domotica-y-tecnologia/

Se creó para artistas, diseñadores, aficionados y cualquier persona interesada en crear entornos u objetos interactivos.

Arduino permite tomar información del entorno a través de sus pines de entrada de toda una gama de sensores y puede afectar aquello que lo rodea controlando, luces, motores, etc.

El microcontrolador de la placa Arduino se programa mediante el lenguaje de programación Arduino (basado en Wiring) y el entorno de desarrollo Arduino (basado en Processing).

Los trabajos hechos con Arduino se pueden ejecutar sin ser necesaria la conexión a un computador, tienen esa posibilidad y también se pueden comunicar con diferentes tipos de software.

[4] David Cuartielles "Proyectos Arduino" Intenet (http://proyectoarduino.wordpress.com/%C2%BFque-es-arduino/)

“Las placas pueden ser hechas a mano o comprarlas montadas de fábrica; el software puede ser descargado y es gratuito. Los ficheros de diseño de referencia CAD están disponibles bajo licencia abierta, así pues se es libre de adaptarlos a las necesidades”[5].

3.6.2 HARDWARE DE ARDUINO

Placas de Arduino hay varias, sin embargo, la actual es la Uno, que usa el microcontrolador Atmel Atmega328 y el Duemilanove. La versión anterior utiliza Diecimila y las primeras unidades con Duemilanove usaban el Atmel Atmega168 y las más antiguas utilizaban el Atmega8.

El módulo GPS para Arduino es un pequeño circuito electrónico que se pude conectar a un módulo Arduino para conseguir la posición y altitud, además la velocidad, la fecha y la hora en UTC (Tiempo Universal Coordinado).

El módulo ensamblado es Vincotech A1080-B que utiliza el estándar NMEA y SIRF III, protocolos (www.nmea.org) para transmitir los datos de posición a través del puerto serie.

El módulo GPS para Arduino es un complemento perfecto para el desarrollo de aplicaciones de geolocalización. Puede ser utilizado con muchos software de navegación compatible con NMEA.

[5] http://www.forocoches.com/foro/showthread.php?t=2884022



Figura 2.9 GPS EM406

Adafruit. “EM406 GPS MODULE” [Fotografía]. Tomado de: http://www.adafruit.com/products/99

3.6.2.14 PLACAS DE E/S

- UNO: es la última revisión de la placa Arduino USB básica. A diferencia de las versiones anteriores, ésta no usa el chip FTDI USB-serie, en este caso Arduino Uno, usa Atmega8U2 programado como un convertidor de USB a serie.

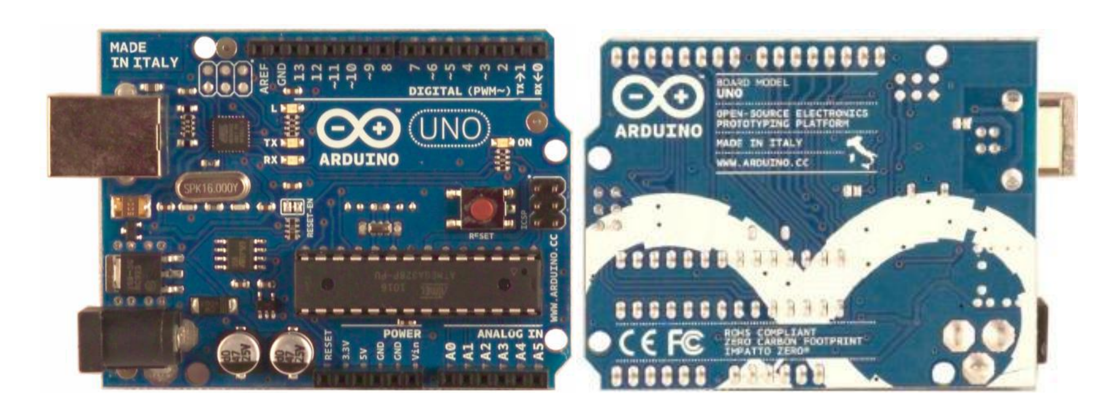


Figura 2.10 Arduino Uno

Seguridad Doméstica. “Domótica con Arduino”. [Fotografía] (2012). Tomado de: http://www.livemodern.org/domotica-con-arduino-domotica-y-tecnologia/

- DUEMILANOVE: traduce 2009 en italiano, éste fue el año cuando el producto salió al mercado. Es el más conocido dentro de las series de placas con USB.

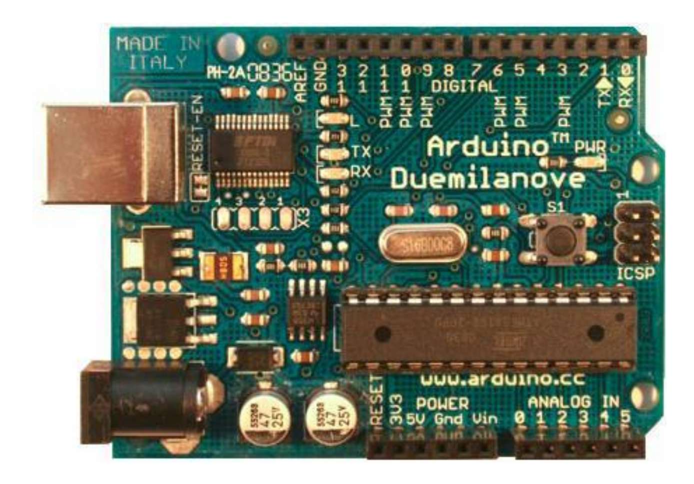


Figura 2.11 Arduino Duemilanove

Seguridad Doméstica. “Domótica con Arduino”. [Fotografía] (2012). Tomado de: http://www.livemodern.org/domotica-con-arduino-domotica-y-tecnologia/

- DIECIMILA: fue la primera versión de Arduino en integrar el chip FTDI para convertir de USB a serie y alimentarse con la energía proporcionada por la salida USB del computador.



Figura 2.12 Figura Diecimila

Seguridad Doméstica. “Domótica con Arduino”. [Fotografía] (2012). Tomado de: http://www.livemodern.org/domotica-con-arduino-domotica-y-tecnologia/

2.6.1.2 SHIELDS ARDUINO

Son las placas impresas que se pueden conectar encima de la placa Arduino, con el fin de que cumpla más funciones y amplíe sus capacidades, sobreponiendo una encima de la otra.

Los diseños de las shields son sencillos y generalmente su código es abierto y son publicados de forma libre.

Las shields más conocidas son:

- Arduino Ethernet Shield
- Arduino microSD Shield
- Arduino Celular Shield SM5100B
- Arduino GPS Shield

2.6.1.3 ARDUINO GPS SHIELD

Un GPS puede registrar todo el trayecto de un vehículo, por lo tanto éste es de fácil uso con la ayuda de Arduino.

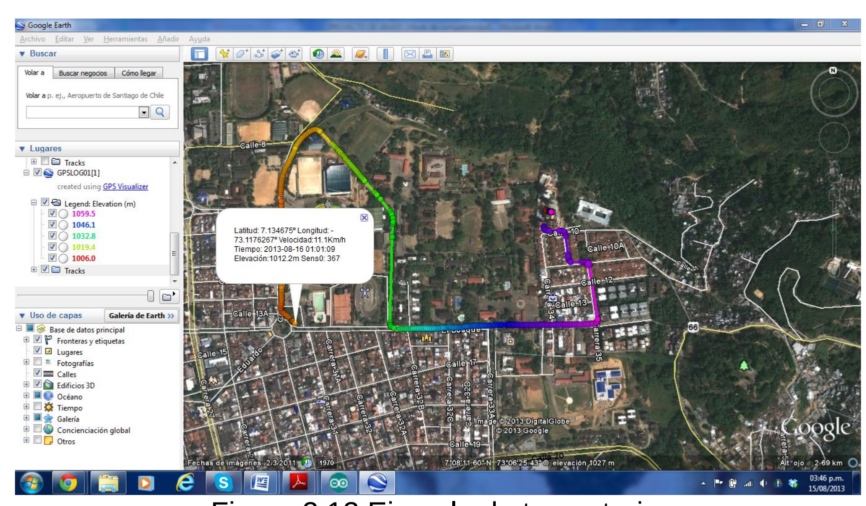


Figura 2.13 Ejemplo de trayectoria.

MANTILLA, Yeidson. “Recorrido” [Fotografía]. (2013).

El Arduino GPS Shield, permite que una placa Arduino se conecte a un sistema GPS, a través de un conector específico para un módulo receptor GPS EM-406.



Figura 2.14 Shield GPS
Arduino Shield List. “Sparkfun GPS Shield” [Fotografía]. (2011). Tomado de: http://shieldlist.org/sparkfun/gps

Con el módulo GPS, un GPS Shield y un Arduino, se podrán obtener datos de la hora y la posición de un módulo GPS.

GPS, permite localizar cualquier objeto que utilice esta tecnología. Éste utiliza tres satélites como mínimo, calculando la longitud y la latitud para ubicar cualquier objeto. En este sentido, se analizará los datos que salen por el pin TX del módulo GPS.

2.6.1.4 ARDUINO SHIELD MICROSD

En ésta se puede conectar la tarjeta microSD de alta capacidad para almacenar los datos necesarios.

La shield microSD está compuesta por un zócalo para tarjeta microSD, junto con un led indicador que indica la alimentación y botón de reset.



Figura 2.15. Módulo Arduino y ShieldSD

Arduino Shield List. "Sparkfun GPS Shield" [Fotografía]. (2011). Tomado de: http://shieldlist.org/sparkfun/gps

2.6.2 ARDUINO UNO

Es una placa con microcontrolador basado en el Atmega328. Está compuesto por 14 pines con entradas y salidas digitales (6 pueden ser usadas como salidas PWM), 6 entradas análogas, un cristal oscilador a 16Mhz, conexión USB, entrada de alimentación DC, una cabecera ICSP y un botón de reset. Se conecta a un computador a través del cable USB para que le genere energía o por el contrario se puede utilizar una batería para trabajar.

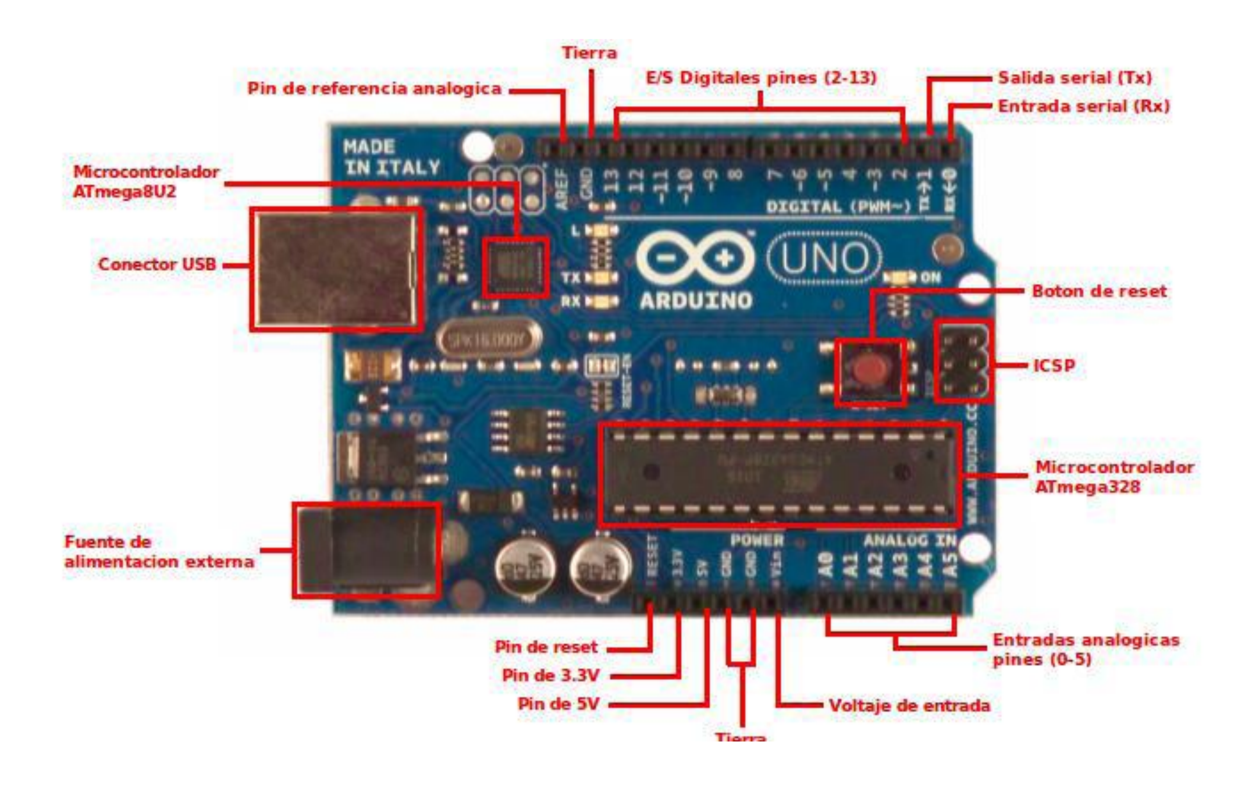


Figura 2.16 Arduino Uno y sus componentes

Alchingadazo. "Iniciando con Arduino: Estructura Del Programa". [Fotografía]. (2012). Tomado de: http://www.alchingadazo.com/

Microcontrolador	Atmega328
Voltaje de funcionamiento	5V
Voltaje de entrada (recomendado)	7-12V
Voltaje de entrada (límite)	6-20V
Pines E/S digitales	14 (6 proporcionan salida PWM)
Pines de entrada análogos	6
Intensidad por pin	40Ma
Intensidad en pin 3.3V	50Ma
Memoria Flash	32KB (2KB las usa el gestor de arranque bootloader)
SRAM	2KB
EEPROM	1KB
Velocidad de reloj	16MHz

Tabla 2.2 Tabla de características

MELGOZA, Jhonatan. "Conociendo Arduino la placa – Empieza a realizar tus proyectos". [Fotografía] (2013).

Tomado de: http://jonathanmelgoza.com/blog/conociendo-arduino-placa/

2.6.2.1 ENERGÍA

Arduino Uno puede ser alimentado a través de la conexión USB o una fuente de alimentación externa DC. La fuente de alimentación se selecciona automáticamente.

La fuente de alimentación externa, es decir que no sea USB, puede ser un adaptador de pared o una batería.

Arduino Uno puede trabajar con una alimentación externa de 6V a 20V. Si se suministra un voltaje inferior a 7V el pin de 5V puede proveer menos de 5V y como consecuencia se puede dar una inestabilidad en la placa; por el contrario si se utilizan más de 12V los reguladores de voltaje se pueden recalentar y quemar la placa; en este sentido entonces, lo recomendado es utilizar una energía entre 7V a 12V.

2.6.2.2 MEMORIA

El Atmega328 tiene una capacidad de 32KB de memoria flash para almacenar códigos (con 0,5KB, que es utilizado por el gestor de arranque); además dispone de 2KB de memoria SRAM y 1KB de EEPROM, en donde se puede acceder para leer o escribir.

2.6.2.3 ENTRADAS Y SALIDAS

Cada uno de los 14 pines digitales de Arduino Uno pueden usarse como entradas o salidas, haciendo utilización de las funciones pinMode (), digitalWrite () y digitalRead (). Tanto las entradas como las salidas funcionan a 5V. Cada uno de los pines pueden proporcionar o recibir una intensidad máxima de 40Ma y tienen una resistencia interna pull up (desconectada por defecto) de 20 a 50 KΩ.

Arduino Uno tiene 6 entradas análogas, y cada una de ellas proporciona una resolución de 10 bits (1.024 valores diferentes). Por defecto se deben medir de tierra a 5V.

2.6.2.4 COMUNICACIÓN

El Arduino Uno permite una serie de comunicaciones con el computador, otro Arduino y otros microcontroladores. El Atmega328 ofrece comunicación a través de UART TTL (5V), que está disponible en los pines digitales 0 (RX) y 1 (TX). El microcontrolador Atmega8U2 que está integrado en la placa, canaliza la comunicación serie a través del USB y proporciona un puerto serie virtual en el computador. El software de Arduino trae un monitor de puerto serie que permite enviar y recibir información de la placa Arduino. Los leds RX y TX del Arduino parpadearán cuando éstos detecten comunicación que será transmitida mediante el Atmega8U2 y la conexión USB (no parpadeará si se usa la comunicación serie a través de los pines 0 y 1).

La librería SoftwareSerial permite la comunicación en serie por cualquiera de los pines diguitales del Uno.

El Atmega328 es compatible con la comunicación I²C (TWI) y SPI. El software de Arduino incluye una librería Wire para simplificar el uso del bus I²C.

2.6.2.5 PROGRAMACIÓN

El Arduino tiene su propio software y de esta manera se puede programar.

El Atmega328 y el Atmega168 en las placas Arduino Duemilanove tienen incluido un gestor de arranque (bootloader), lo que permite cargar un nuevo código sin ser necesario un programador hardware externo. En este caso se realiza la comunicación utilizando el protocolo STK500 original, o por el contrario no se utiliza el gestor de arranque y se puede programar directamente el microcontrolador a través del puerto ISCP (In Circuit Serial Programming).

2.6.2.6 REINICIO AUTOMÁTICO (SOFTWARE)

El Arduino está diseñado de tal manera que se puede reiniciar por software desde el computador donde esté conectado. "*Una de las líneas de control de flujo (DTR) del FT232RL está conectada a la línea de reinicio del Atmega328 o Atmega168 a través de un condensador de 100 naanofaradios. Cuando la línea se pone a LOW (Ov), la línea de reinicio también se pone a LOW el tiempo suficiente para reiniciar el chip. El software de Arduino utiliza esta característica para permitir cargar los sketches con sólo apretar un botón del entorno. Dado que el gestor de arranque tiene un lapso de tiempo para ello, la activación del DTR y la carga del sketch se coordinan perfectamente*"[6].

2.6.2.7 PROTECCIÓN CONTRA SOBRECARGAS EN EL PUERTO USB

El Arduino Uno tiene incorporado un multifusible reiniciable, con el fin de proteger la conexión USB del computador de cortocircuitos o sobrecargas. Aunque cada computador tiene su propia protección interna, el multifusible le proporciona una capa extra de protección. Si se detectan más de 500Ma en el puerto USB, el multifusible corta la conexión hasta que el corto o la sobrecarga desaparezca.

---

[6] Arduino. "Arduino Duemilanove": http://arduino.cc/es/Main/arduinoBoardDuemilanove

2.6.2.8 CARACTERÍSTICAS FÍSICAS

La longitud y amplitud de la placa Arduino es de 2,7 y 2,1 pulgadas, tiene un conector USB y la conexión de alimentación sobresaliendo de estas dimensiones. También trae tres agujeros para la fijación con tornillos que permiten colocar la placa en superficies y cajas.

2.6.2.9 SOFTWARE PARA ARDUINO

Arduino como es de código abierto, permite escribir el código y cargarlo a la placa de E/S de manera fácil. Funciona con Windows, Mac IOS, Linux y Android. El entorno está escrito en Java y basado en Processing, avr-gcc y otros programas también de código abierto.

2.6.2.10 ENTORNO ARDUINO

También es conocido como IDE para Arduino. Se constituye por un editor de textos para escribir el código, un área de mensajes, una consola de textos, una barra de herramientas con botones para las funciones comunes y una serie de menús. De esta forma, permite la conexión con el hardware de Arduino para cargar los programas y comunicarse con ellos.

El Arduino utiliza el software denominado sketch (programa) para escribir. Estos programas son escritos en el editor de texto, con funciones como cortar, pegar y buscar/reemplazar texto. En el área de mensaje mientras se carga el programa se muestra información y se señalan los errores. Se muestra el texto de salida para el entorno de Arduino incluyendo mensajes de error completos y otras informaciones. En la barra de herramientas se verifica el proceso de carga, apertura, creación y guardado de programas y la monitorización serie.

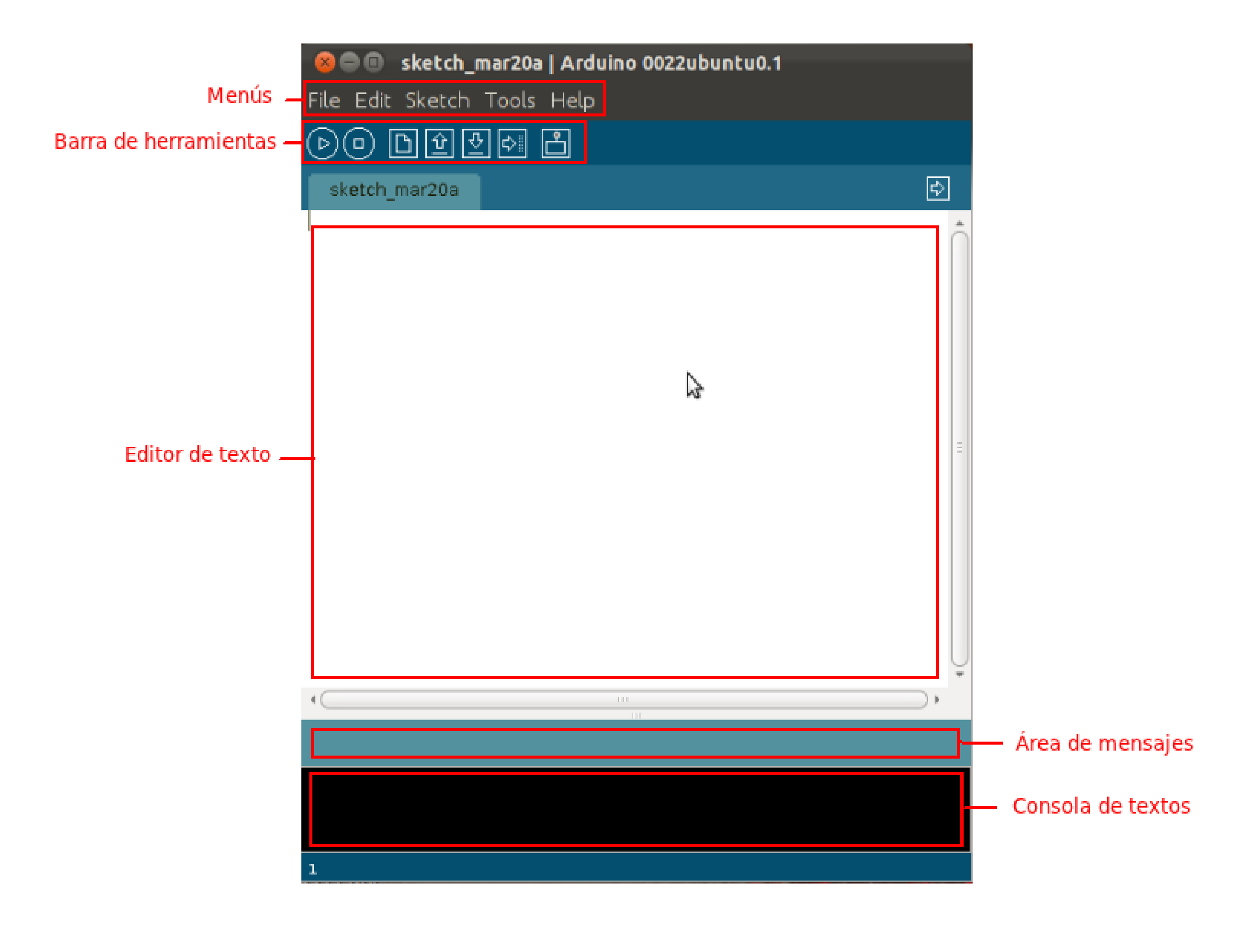


Figura 2.17 IDE para Arduino. Versión 0022

Alchingadazo. "Iniciando con Arduino: Estructura Del Programa". [Fotografía]. (2012). Tomado de: http://www.alchingadazo.com/

2.6.2.11 BARRA DE HERRAMIENTAS

En el IDE del Arduino se pueden encontrar una serie de herramientas, cada una con diferentes funciones como se explica a continuación:

- Verify/Compile: Chequea el código en busca de errores.
- Stop: Finaliza la monitorización serie y oculta otros botones.
- New: Crea un nuevo sketch.
- Open: Presenta un menú de todos los programas sketch "sketchbooks" (librería sketch).

- Save: Guarda el programa sketch.
- Upload to I/O Board: Compila el código y lo vuelca en la placa E/S Arduino.
- Serial Monitor: Inicia la monitorización serial.

2.6.2.12 MENÚS

Arduino tiene una serie de menús, que sólo se habilitan de acuerdo a la acción que se esté efectuando en el momento. Éstos son:

MENÚ	DESCRIPCIÓN
File	Búsqueda de archivo creado.
New	Crear archivo nuevo.
Open	Abrir archivos.
Sketchbook	Abre los sketch creados.
Examples	Abre los sketch que están de ejemplo, al descargar el IDE para Arduino.
Close	Cierra el IDE.
Save	Guarda los cambios del sketch.
Save As	Guardar con un nombre y en un lugar específico.
Upload to I/O board	Compila el código y lo vuelca en la placa E/S Arduino, esta opción también se encuentra en la barra de herramientas.
Page Setup	Ingresa a configuración de vista en el IDE.
Print	Imprime el código.
Preferences	Permite modificar opciones del editor, sobretodo almacenar los sketch creados.
Quit	Cierra la pantalla del IDE
Edit	Edita el código para realizar cambios.
Undo	Deshace un paso antes de la escritura del código.
Redo	Avanza un paso después.
Cut	Corta una línea o varias del código.
Copy	Copia una línea o varias del código.
Copy for fórum	Copia al portapapeles el texto del sketch seleccionado, el texto posee el

	formato usado en los foros.
Copy As HTML	Copia al portapeles el texto del sketch seleccionado, el texto tiene el formato HTML para páginas web.
Paste	Pega los copy y cut.
Select All	Selecciona todo el código que se encuentre dentro del IDE.
Comment/Uncomment	Inicia y finaliza los comentarios en el sketch.
Increase Indent	Aumenta el guión y sangría.
Decrease Indent	Disminuye el guión y sangría.
Find	Encuentra un archivo.
Find Next	Continúa la secuencia de búsqueda de archivos.
Sketch	Programa o código finalizado.
Verify/compile	Verifica los errores del programa.
Stop	Detiene la verificación de los errores.
Show Sketch Folder	Abre la carpeta de programas.

Tabla 2.2. Tabla de menús

2.6.2.13 TOOLS

Auto Format: tabula el código dándole formato para que él mismo a la hora de compilar sea más estético y libre de errores, por espacios indebidos en la escritura.
Fix Encoding&Reload: permite actualizar la versión del sketch, con el fin de que sea compatible en algunas de sus líneas, aunque en algunos casos es necesario acudir a la página de Arduino para verificar diferentes cambios en comandos y así reemplazarlos manualmente en el código.
Serial Monitor: permite acceder a la información generada por la placa Arduino y/o las shields conectadas a él.
Board: listado de todos los módulos Arduino diseñados hasta el día de hoy, para seleccionar el módulo a trabajar.
Serial Port: Contiene todos los dispositivos seriales (virtuales o reales).
Verify: compila el código escrito.
Upload: carga el código en el Atmega328.

2.6.2.14 ESTRUCTURA

El Arduino tiene una estructura básica del lenguaje de programación, que se compone de dos funciones, que encierran bloques que contienen instrucciones y son:

- Voidsetup()
- Voidloop()

Setup() inicialización
Siempre que se vaya a escribir un programa se debe establecer un voidsetup, con el fin de iniciar variables, estado de los pines e inicialización de comunicación, esto sólo se hace una vez después de que se conecte la placa a la fuente de poder.

Loop() *bucle*
Luego de crear la función **setup**(), la cual inicializa y prepara los valores iniciales, la función **loop**() hace justamente lo que su nombre sugiere, por lo tanto se ejecuta continuamente, permitiéndole al programa variar y responder, leyendo entradas, activando salidas, etc. Esta función es el núcleo de todos los programas de Arduino y hace la mayor parte del trabajo.

Sintaxis
La sintaxis del lenguaje es muy parecida a la de C y C++, manteniendo las mismas estructuras, que son las siguientes:

- Funciones
- Llaves
- Punto y coma
- Bloques de comentarios
- Comentarios de línea

Funciones
Es un bloque de código que tiene un nombre y un grupo de declaraciones que sse ejecutan cuando se llama a la función. Se puede hacer uso de funciones integradas como **void**setup() y **void**loop() o escribir nuevas.
Las funciones se escriben para ejecutar tareas repetitivas y reducir el desorden en un programa. En primer lugar se declara el tipo de la función, que será el valor retornado por la función (int, void…). A continuación del tipo, se declara el nombre de la función y, entre paréntesis, los parámetros que se pasan a la función.

Llaves {}
Definen el comienzo y el final de bloques de función y bloques de declaraciones como voidloop() y sentencias for e if. Las llaves deben estar balanceadas (a una llave de apertura {debe seguirle una llave de cierre}). Las llaves no balanceadas provocan errores de compilación.

El entorno Arduino incluye una práctica característica para chequear el balance de llaves. Sólo selecciona una llave y su compañera lógica aparecerá resaltada.

Punto y coma (;)
Un punto y coma debe usarse al final de cada declaración y separa los elementos del programa. También se usa para separar los elementos en un bucle for.
Olvidar un punto y coma al final de una declaración producirá un error de compilación.

Bloques de comentarios /*…*/
Los bloques de comentarios, o comentarios multilínea, son áreas de texto ignoradas por el programa y se usan para grandes descripciones de código o comentarios que ayudan a otras personas a entender partes del programa. Empiezan con /* y terminan con */ y pueden abarcar múltiples líneas. Los comentarios son ignorados por el programa y no ocupan espacio en memoria.

Comentarios de línea //
Comentarios de una línea empiezan con // y terminan con la siguiente línea de código. Como los bloques de comentarios son ignorados por el programa, no toman espacio en memoria. Comentarios de una línea se usan a menudo después de declaraciones válidas para proporcionar más información sobre qué lleva la declaración o proporcionar un recordatorio en el futuro.

Variables
Sirven para llamar y almacenar valores numéricos, estos números pueden cambiar dependiendo la necesidad del programador.

Declaración de variable
Para que una variable pueda ser identificada en el programa debe ser primero declarada, definiendo su tipo de valor, como long, float, int, etc. Esto se hace sólo una vez, pues no es necesario repetir la declaración dentro del programa.

Ejemplo de Estructura:

//comienza la nvía uración

```
void setup() {
```

//configura el pin 13 como de salida

```
pinMode(13, OUTPUT);
```

```
}
```

//termina la   nvía   uración

//comienza el bucle principal del programa

```
void loop() {
```

//   nvía 5V al pin (salida) 13

```
digitalWrite(13, HIGH);
```

//espera 500 ms pin 13 con 5V

```
delay (500);
```

//   nvía 0V al pin (salida) 13

```
digitalWrite(13, LOW);
```

//espera 100 ms pin 13 con 0V

```
delay (100);

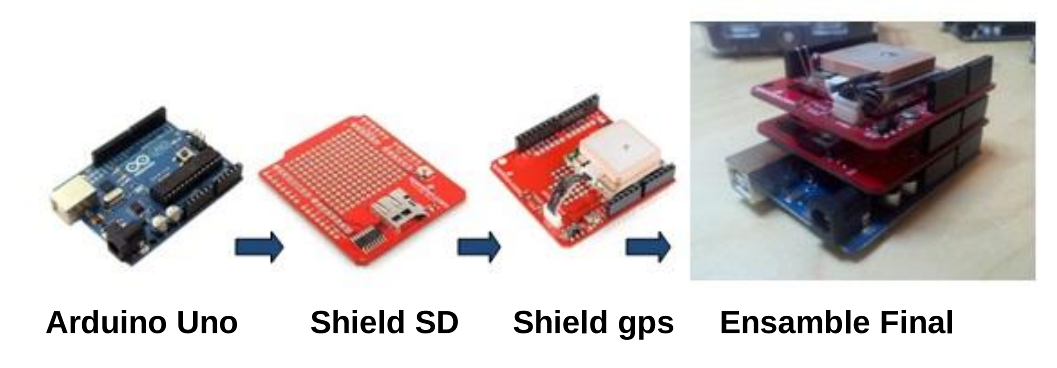
}
```

CAPÍTULO 3
DISEÑO E IMPLEMENTACIÓN

Para el diseño y la implementación del proyecto, son necesarios los siguientes componentes:

3.1 HARDWARE

La familia Arduino es muy numerosa, se empezó por el Arduino Leonardo, pasando por el Duemilanove, hasta llegar el más completo de todos que es el Arduino Uno, debido a su mayor velocidad de transmisión e impresión de datos hasta 115.200 baudios. Esto hace de Arduino Uno, una tarjeta formidable para trabajar datos con un GPS, aunque recibe un dato por segundo, no se le hacen bucles y información es enviada a la tarjeta.



3.1. Diagrama de bloques

3.1.1 ARDUINO UNO

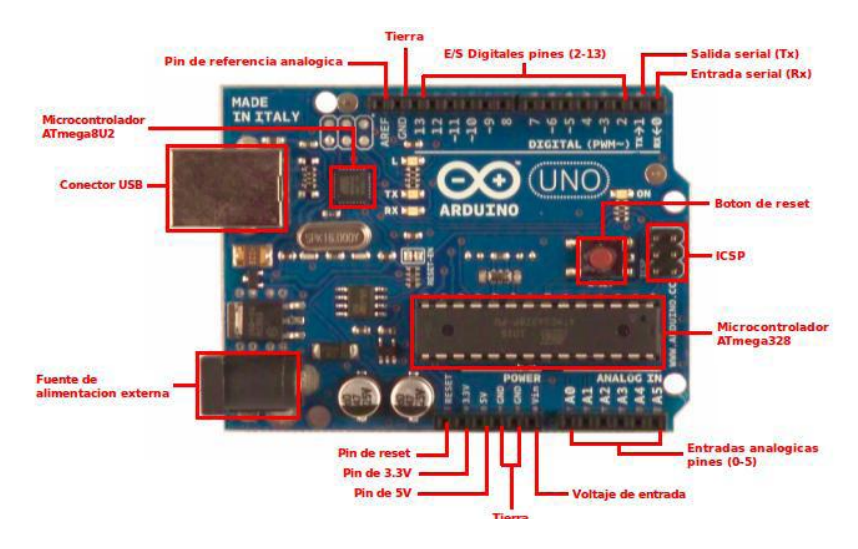


Figura 3.2 Partes de Arduino Uno

Alchingadazo. "Iniciando con Arduino: Estructura Del Programa". [Fotografía]. (2012). Tomado de: http://www.alchingadazo.com/

Pines de alimentación:

Pines de alimentación	Descripción
VIN	el voltaje se puede proporcionar a través de este pin, o, si se está alimentando a través de la conexión externa de 2,1mm, se puede acceder a ella a través de este pin (7 a 12V).
5V	es el pin de salida de voltaje estabilizado, éste es proporcionado por el VIN mediante un regulador integrado al Arduino Uno o directamente de la USB. No es aconsejable, debido a que

	el suministro de tensión a través de los 5V o 3.3V pins no pasa por el regulador.
3V3	es una fuente de alimentación de 3.3V, que es generada por el regulador que viene con el Arduino. Soporta una corriente máxima de 50Ma.
GND	Pines de toma a tierra.
IOREF	Proporciona la tensión con la que trabaja el microcontolador. Una placa configurada puede leer el voltaje pin IOREF y de esta manera seleccionar la fuente de alimentación más adecuada como también habilitar traductores de voltaje en las salidas para trabajar con los pines 5V ó 3.3V.

Tabla 3.1. Pines de alimentación – Arduino Uno

Algunos de los pines tienen una función específica:

PINES CON FUNCIÓN ESPECÍFICA	DESCRIPCIÓN
Serie: pin 0 (RX) y 1 (TX)	Es utilizado para recibir (RX) y transmitir (TX) a través del puerto serie TTL. Los pines están conectados en paralelo a los pines correspondientes de la Atmega8U2 y a los pines RXD y TXD del Atmega.
Interrupciones externas: 2 y 3.	Pueden ser configurados para activar una interrupción en un valor LOW (0V), en límites de subida o bajada (cambio de LOW a HIGH o viceversa), o en cambios de valor.

PWM: 3, 5, 6, 9, 10, 11.	Proporciona una salida de PWM (Pulse Wave Modulation = modulación por onda de pulso) de 8 bits de resolución con valores de 0 a 255. Se los identifica por el símbolo ~ en la placa Arduino.
SPI: 10 (SS), 11(MOSI), 12 (MISO), 13 (SCK).	Estos pines proporcionan comunicación SPI, ya que a pesar de que el hardware lo proporcione actualmente no está incluido en el lenguaje Arduino.
LED: 13.	En este pin hay un led conectado al pin digital 13. Cuando este pin tiene un valor alto, es decir HIGH (5V), el led se enciende y cuando el pin tiene un valor bajo, o sea LOW (0V), el led se apaga.

Tabla 3.2. Pines con función específica – Arduino Uno

Algunos pines tienen funciones especializadas:

PINES CON FUNCIONES ESPECIALIZADAS	DESCRIPCIÓN
I^2C: pin 4 (SDA) y pin 5 (SCL).	Soporte del protocolo de comunicaciones I^2C (TWI) usando la librería Wire.

Tabla 3.3. Pines con funciones espcializadas – Arduino Uno

Otros pines de Arduino Uno:

OTROS PINES DE ARDUINO	DESCRIPCIÓN
AREF	Voltaje de referencia para las entradas análogas. La función analogReference () devolverá un valor de 1023 para aquella tensión de entrada que sea igual a la tensión de referencia. El valor del voltaje debe estar en el rango de 0

	a 5V.
AREF:RESET	Suministra un valor LOW (0V) para reiniciar el microcontrolador. Generalmente se utiliza para agregar un botón de reinicio a los Shields que no permiten acceso a la placa.

Tabla 3.4. Otros pines – Arduino

3.1.2 SHIELD SD

El GPS es una herramienta bastante útil y fácil de usar con la ayuda de Arduino. Se puede hacer un registro de un recorrido de un viaje en carro o demacar una ruta por ejemplo, es posible con un GPS.



Figura 3.3 Demo recorrido GPS
Nissitech. "GPS Vehicle Tracking". [Fotografía]. Tomado de: http://nissitech.net/info_gps.php

3. 1.3 GPS EM406

Para obtener datos como la hora y la posición del módulo GPS, se requiere de un módulo GPS, la placa de GPS, el Arduino y una placa SD.



Figura 3.4 Shield GPS
Arduino Shield List. “Sparkfun GPS Shield” [Fotografía]. (2011). Tomado de: http://shieldlist.org/sparkfun/gps

El GPS puede mostrar una ubicación de forma inmediata, con éste se pueden hacer seguimientos, tomar registros, también precisa el tiempo, es decir, va más allá de mostrar la latitud y la longitud.

Para comenzar a trabajar, lo primero que se debe realizar es el montaje de la placa GPS, luego se elige un módulo GPS que funcione con la placa GPS, en este caso es el conector para un EM-406.

Las siguientes son las opciones de configuración para la placa GPS:

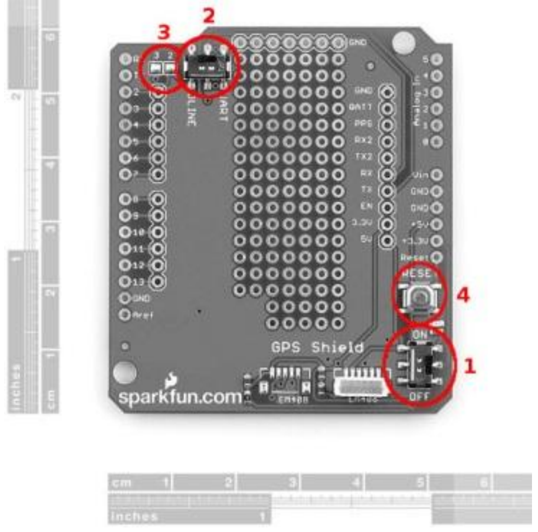


Figura 3.5 Configuracion Shield GPS

Sparkfun. “GPS Shield”. [Fotografía]. (2010). Tomado de: https://www.sparkfun.com/tutorials/173

Hay un interruptor on/off, un botón de reset, un botón UART y Dline que serán explicados a continuación:

1. **Interruptor de encendido:** controla la alimentación del módulo GPS

2. **Interruptor de selección de UART:** Si se selecciona la UART, el módulo GPS es conectado a los pines digitales Arduino 0 y 1. Si se selecciona Dline, el GPS se conecta a los pines digitales 2 y 3 (por defecto) o potencialmente cualquier otro pin digital. Dline debe ser seleccionado con el fin de cargar el código. La razón de esto es porque la selección UART utiliza las mismas líneas que se utilizan para la programación. Si se selecciona la UART y carga el código, es probable obtener errores en el IDE de Arduino.

3. **Botón de reset:** es utilizado en caso de bloqueos que ocasionen demora en la obtención de datos, bien sea por fallas de alimentación o por bucles. Se oprime una sola vez y el programa vuelve al inicio.

Con el módulo GPS se reciben datos que siguen el protocolo NMEA que corresponde a National Marine Electronics Asociation y tienen la siguiente estructura:

$GPRMC,054235.000,A,4522.0289,N,00724.5210,E,0.39,65.46, 070613,,,A*44

Empiezan por "$" y acaban en un "A*" seguido de dos números, éste es el checksum para poder saber si el dato recibido es correcto. Los datos se separan por comas y el primero es el tipo de transmisión, en este caso el GPRMC (o RMC), uno de los más usados, que por ejemplo no incluye el valor de altitud, que si se incluye en el GPGGA. Todo el protocolo se enlaza en la sección de descargas. Se revisan los datos para entenderlos o interpretarlos:

- **044235.000** hora GMT (05:42:35)
- **A** dato fijado y correcto
- **4522.0289** long (45º 22.0289´)
- **N** Norte
- **00724.5210** es la lat (7º 24.5210´)
- **E** Oeste
- **0.39** velocidad (nudos)
- **65.46** grados
- **070613** fecha (7 de junio del 2013)

Para lograr la recepción de datos se requiere del módulo GPS, en este caso es el EM-406A, es uno de los más utilizados, es potente, pequeño y tiene una antena integrada. De esta forma logra obtener datos cada segundo y se alimenta con 5V y su comunicación es por serie.

El GPS Shield sólo se enchufa y se monta al Arduino, ya que tiene un microconector. En esta ocasión se configuran los dos mini interruptores de la Shield como ON y Dline. El módulo permanece encendido por lo tanto envía datos por los pines digitales 2 y 3.

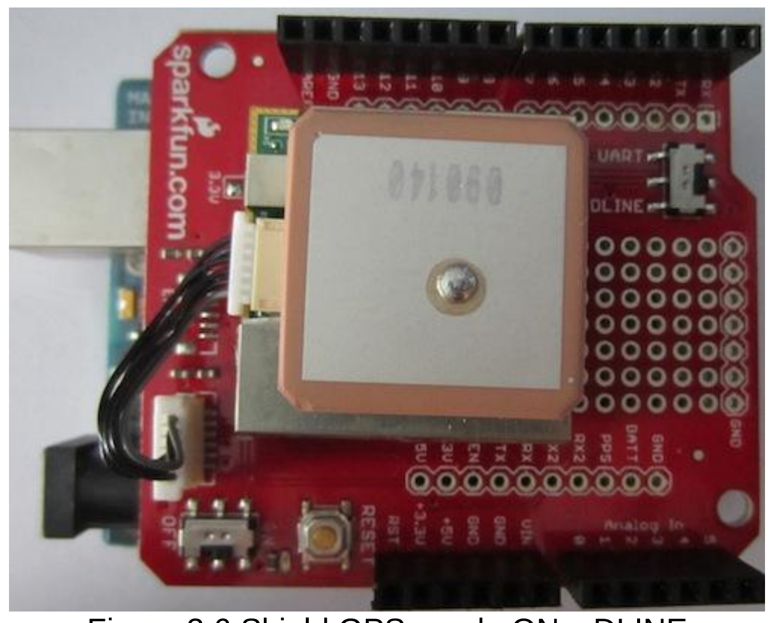
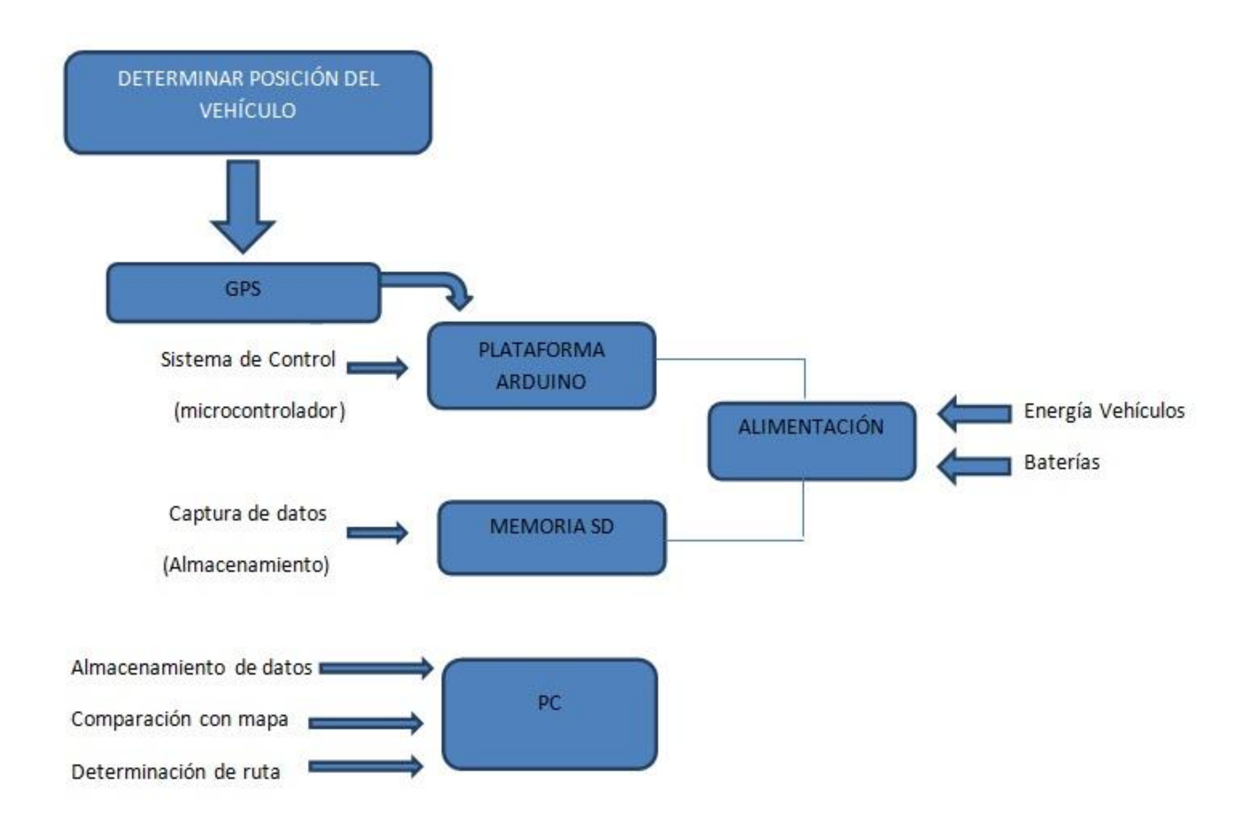


Figura 3.6 Shield GPS, modo ON y DLINE

Homotix. “GPS SHIELD” [Fotografía]. Tomado de: http://www.homotix.it/catalogo/arduino-shield/gps-shield

Es necesario incluir dos librerías en el IDE de Arduino, TinyGPS y NewSoftSerial; para esto se dejan en una carpeta que queda adjunta; éstas deben ser copiadas en el directorio Libraries, después se abre el código:



3.7 Diagrama de flujo

Definir qué pines se van a utilizar en el Arduino para comunicarse con el GPS. En este caso, el pin TX del GPS se conectará al pin RX de Arduino que es el pin 3.

```
#define RXPIN 2
#define TXPIN 3
//Establezca este valor igual a la velocidad de transmisión del GPS (baudios)
#define GPSBAUD 4800
```

Cree una petición de la biblioteca y TinyGPS y el módulo GPS

```
TinyGPS gps;
```

Inicializar la biblioteca NewSoftSerial en los pines que ha definido anteriormente.

```
NewSoftSerial uart_gps(RXPIN, TXPIN);
```

Aquí es donde se debe declarar las funciones que serán usadas por la biblioteca TinyGPS.

```
void getgps(TinyGPS &gps);
```

Se debe inicializar dos puertos serie, el puerto serial estándar (hardware) para comunicarse con el programa y otro puerto serial para comunicación con el GPS.

```
void setup()
{
```

Este es el serial rate, es decir, la velocidad con que los datos se verán en la pantalla terminal.

```
.
 Serial.begin(115200);
```

Seleccione el baud rate del GPS.

```
uart_gps.begin(GPSBAUD);
Serial.println("");
Serial.println("Inicio rápido de GPS, código ejemplo");
Serial.println("       ...esperando información...           ");
Serial.println("");
}
```

Este es el bucle principal del código. Todo lo que hace es comprobar los datos en El pin RX del ardiuno, se asegura de que los datos son válidos y son sentencias NMEA.

```
void loop()
{
  while(uart_gps.available())     // Mientras hayan datos en el pin RX...
  {
      int c = uart_gps.read();    // Carga los datos dentro de una variable...
      if(gps.encode(c))      // Si hay una sentencia válida...
      {
        getgps(gps);         // éste graba los datos.
      }
  }
}
```

La función getgps recibe e imprime los valores que se quieren.

```
void getgps(TinyGPS &gps)
{
```

Para obtener todos los datos en varialbes que se puede utilizar en el código, Todo lo que se necesita hacer es definir las variables y consultar el objetivo para la obtención de datos.

Definir las variables que se usarán.

```
float latitude, longitude;
```

LLamar las funciones

```
gps.f_get_position(&latitude, &longitude);
```

Imprimir las variables latitud y longitud.

```
Serial.print("Lat/Long: ");
Serial.print(latitude,5);
Serial.print(", ");
Serial.println(longitude,5);
```

Lo mismo sucede con la fecha y la hora.

```
  int year;
  byte month, day, hour, minute, second, hundredths;
  gps.crack_datetime(&year,&month,&day,&hour,&minute,&second,&hundredths);
```

Imprimir fecha y hora.

```
Serial.print("Date: "); Serial.print(month, DEC); Serial.print("/");
Serial.print(day, DEC); Serial.print("/"); Serial.print(year);
Serial.print("  Time: "); Serial.print(hour, DEC); Serial.print(":");
Serial.print(minute, DEC); Serial.print(":"); Serial.print(second, DEC);
Serial.print("."); Serial.println(hundredths, DEC);
```

Imprime los valores de altitud y rumbo.

```
Serial.print("Altitude (meters): "); Serial.println(gps.f_altitude());
```

Lo mismo pasa para el curso.

```
Serial.print("Course (degrees): "); Serial.println(gps.f_course());
```

Y lo mismo para la velocidad.

```
Serial.print("Speed(kmph): "); Serial.println(gps.f_speed_kmph());
Serial.println();
```

Imprime las estadísticas sobre las sentencias.

```
unsigned long chars;
unsigned short sentences, failed_checksum;
gps.stats(&chars, &sentences, &failed_checksum);
//Serial.print("Failed Checksums: ");Serial.print(failed_checksum);
//Serial.println(); Serial.println();
}
```

Se carga en el Arduino, se abre el puerto serie y en el momento que el GPS fije la posición, el led GPS parpadea consiguiendo la ubicación.

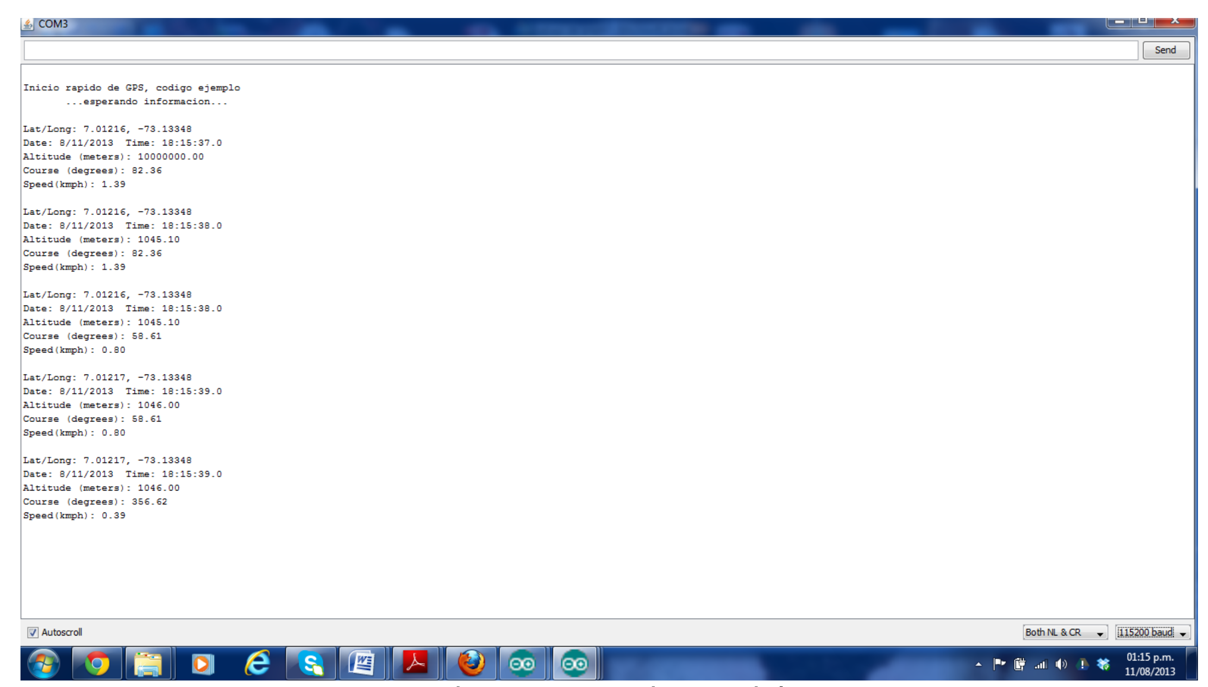


Figura 3.8 Monitor Serial
ANTOLINES, Jorge. “Monitor Serial”. [Fotografía] (2013).

Ahora se desconecta el puerto serial y se alimenta de energía el dispositivo con una batería externa, en este caso se usa cuatro pilas AAA de 1.5V cada una. Se le añade un led para monitorear el estado de la escritura en la MicroSD y el GPS. Se usa el receptor GPS a través del puerto serie del Arduino, pines 0 y 1. Se sitúan los interruptores en OFF y UART; OFF apaga el módulo, ya que para programar el Arduino se usa el serial y si se tiene el receptor encendido no se programa adecuadamente.

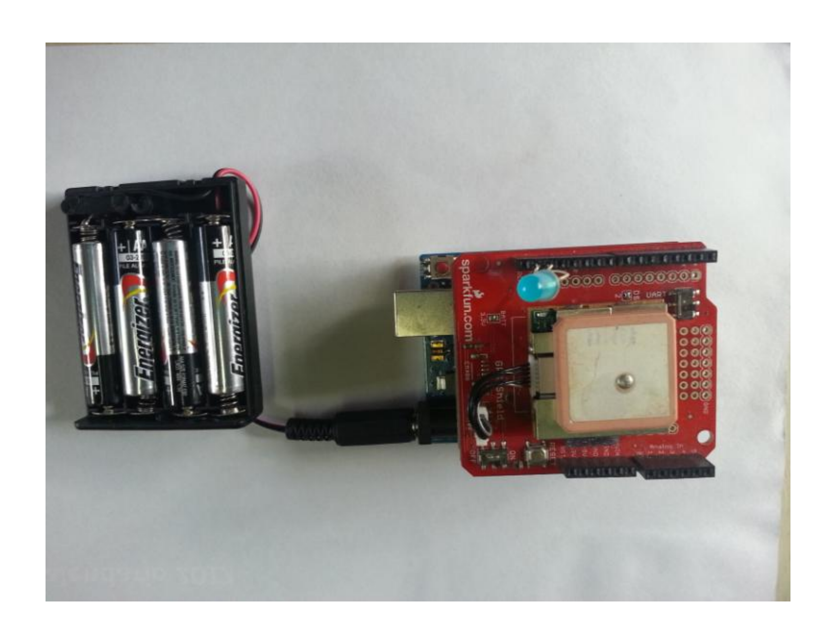


Figura 3.9 Alimentación externa
Homotix. “GPS SHIELD” [Fotografía]. Tomado de: http://www.homotix.it/catalogo/arduino-shield/gps-shield

Se incluye una nueva librería la SDFat (en carpeta adjunta) y se carga el código:



3.10 Diagrama de Bloques

```
uint8_t sensorCount = 1; //numero de sensores analogicos a loguear

// Librerias para la SD
#include <SdFat.h>
#include <SdFatUtil.h>
#include <avr/pgmspace.h>

#define isdigit(x) ( x >= '0' && x <= '9')

//extern uint16_t _end;
```

```
Sd2Card card;
SdVolume volume;
SdFile root;
SdFile f;

#define led1Pin 2         // LED de estado en el pin 7
```

CONFIGURACION DEL MODULO GPS (VER MANUAL DE REFERENCIA NMEA

```
#define LOG_RMC  1      // Datos de localizacion esenciales RMC
#define RMC_ON    "$PSRF103,4,0,1,1*21\r\n"   // Comando que se manda para activar RMC (1 hz)
#define RMC_OFF   "$PSRF103,4,0,0,1*20\r\n"   // Comando que se manda para desactivar RMC

#define LOG_GGA  0      // contains fix, hdop & vdop data
#define GGA_ON    "$PSRF103,0,0,1,1*25\r\n"    // Comando que se manda para activar GCA (1 hz)
#define GGA_OFF   "$PSRF103,0,0,0,1*24\r\n"    // Comando que se manda para desactivar GGA

#define LOG_GSA 0      // satellite data
#define GSA_ON    "$PSRF103,2,0,1,1*27\r\n"    // Comando que se manda para activar GSA (1 hz)
#define GSA_OFF   "$PSRF103,2,0,0,1*26\r\n"    // Comando que se manda para desactivar GSA

#define LOG_GSV  0      // detailed satellite data
#define GSV_ON    "$PSRF103,3,0,1,1*26\r\n"   // Comando que se manda para activar GSV (1 hz)
#define GSV_OFF   "$PSRF103,3,0,0,1*27\r\n"   // Comando que se manda para desactivar GSV

#define LOG_GLL 0      // Loran-compatibility

#define USE_WAAS   1      // Util para conseguir mas precision, pero mas lento y consume mas bateria
#define WAAS_ON     "$PSRF151,1*3F\r\n"        // Comando que se manda para activar WAAS
```

```
#define WAAS_OFF    "$PSRF151,0*3E\r\n"        // Comando que se manda para desactivar WAAS

#define LOG_RMC_FIXONLY 1  // loguear solo cuando se tiene la posicion fijada en el RMC?
uint8_t fix = 0; // dato actual de fijacion de posicion
```

Lee un valor hexadecimal y devuelve su valor decimal

```
uint8_t parseHex(char c) {
    if (c < '0')  return 0;
    if (c <= '9') return c - '0';
    if (c < 'A')  return 0;
    if (c <= 'F') return (c - 'A')+10;
}

uint8_t i;
```

Parpadea el LED si hay un error

```
void error(uint8_t errno) {
  if (card.errorCode()) {
    putstring("Error en la SD: ");
    Serial.print(card.errorCode(), HEX);
    Serial.print(',');
    Serial.println(card.errorData(), HEX);
  }
   while(1) {
     for (i=0; i<errno; i++) {
       digitalWrite(led1Pin, HIGH);
       Serial.println("error");
       delay(100);
       digitalWrite(led1Pin, LOW);
       delay(100);
     }
     for (; i<10; i++) {
       delay(200);
     }
   }
}
```

```
void setup()
{
```

```
  Serial.begin(4800);  //a esta frecuencia funciona el GPS
  pinMode(10, OUTPUT); //esencial para que funcione la microSD shield
  putstring_nl("GPSlogger BricoGeek");
  pinMode(led1Pin, OUTPUT);      // selecciona el pin digital de salida.
   pinMode(13, OUTPUT);
  if (!card.init()) {
    putstring_nl("La inicializacion de la SD ha fallado!");
    error(1);
  }
  if (!volume.init(card)) {
    putstring_nl("No hay particion!");
    error(2);
  }
  if (!root.openRoot(volume)) {
        putstring_nl("No se puede abrir el directorio raiz");
    error(3);
  }
  strcpy(buffer, "GPSLOG00.CSV"); //empezara a escribir en 00, si ya hay un archivo previo incrementa la cuenta
  for (i = 0; i < 100; i++) {
    buffer[6] = '0' + i/10;
    buffer[7] = '0' + i%10;
    if (f.open(root, buffer, O_CREAT | O_EXCL | O_WRITE)) break;
  }

  if(!f.isOpen()) {
    putstring("No se ha podido crear "); Serial.println(buffer);
    error(3);
  }
  putstring("escribiendo en "); Serial.println(buffer);
  putstring_nl("Preparado!");
```

// escritura de la cabecera

```
  if (sensorCount > 6) sensorCount = 6;
  strncpy_P(buffer, PSTR("time,lat,long,speed,date,sens0,sens1,sens2,sens3,sens4,sens5"), 24 + 6*sensorCount);
  Serial.println(buffer);
  // clear print error
  f.writeError = 0;
  f.println(buffer);
  if (f.writeError || !f.sync()) {
    putstring_nl("no se pudo escribir la cabecera!");
```

```
    error(5);
  }

  delay(1000);
```

Aqui se hace la configuracion del gps que definimos mas arriba

```
   putstring("\r\n");
#if USE_WAAS == 1
   putstring(WAAS_ON); // on WAAS
#else
  putstring(WAAS_OFF); // on WAAS
#endif

#if LOG_RMC == 1
  putstring(RMC_ON); // on RMC
#else
  putstring(RMC_OFF); // off RMC
#endif

#if LOG_GSV == 1
  putstring(GSV_ON); // on GSV
#else
  putstring(GSV_OFF); // off GSV
#endif

#if LOG_GSA == 1
  putstring(GSA_ON); // on GSA
#else
  putstring(GSA_OFF); // off GSA
#endif

#if LOG_GGA == 1
 putstring(GGA_ON); // on GGA
#else
 putstring(GGA_OFF); // off GGA
#endif
}
```

//

```
void loop()
{
  //Serial.println(Serial.available(), DEC);
  char c;
```

```
uint8_t sum;
```

lee una linea NMEA del GPS

```
if (Serial.available()) {
  c = Serial.read();
  //Serial.print(c, BYTE);
  if (bufferidx == 0) {
    while (c != '$')
      c = Serial.read(); // espera que nos llegue un $
  }
  buffer[bufferidx] = c;

  //Serial.print(c, BYTE);
  if (c == '\n') {
    //putstring_nl("EOL");
    //Serial.print(buffer);
    buffer[bufferidx+1] = 0; // termina

    if (buffer[bufferidx-4] != '*') {
      // no hay checksum?
      Serial.print('*', BYTE);
      bufferidx = 0;
      return;
    }
```

calcula checksum

```
    sum = parseHex(buffer[bufferidx-3]) * 16;
    sum += parseHex(buffer[bufferidx-2]);
```

comprueba checksum

```
    for (i=1; i < (bufferidx-4); i++) {
      sum ^= buffer[i];
    }
    if (sum != 0) {
      //putstring_nl("error de checksum");
      Serial.print('~', BYTE);
      bufferidx = 0;
      return;
    }
```

tenemos datos!

```
    //Serial.println(buffer);
```

```
    if (strstr(buffer, "GPRMC")) {
```

buscamos si tenemos la posicion fijada

```
      char *p = buffer;
      p = strchr(p, ',')+1;
      p = strchr(p, ',')+1;       // vamos a el tercer dato

      if (p[0] == 'V') {
        digitalWrite(led1Pin, LOW);  // no tenemos fijada la posicion
        fix = 0;
      }
      else {
        digitalWrite(led1Pin, HIGH); // tenemos fijada la posicion
        fix = 1;
      }
    }
```

aqui esperamos si lo hemos configurado para que solo coja datos con la posicion fijada

```
#if LOG_RMC_FIXONLY
    if (!fix) {
        Serial.print('_', BYTE);

        bufferidx = 0;
        return;
    }
#endif

    Serial.println();
    Serial.print("Secuencia NMEA recibida: ");
    Serial.print(buffer);
```

buscando los datos

```
    char *p = buffer;
    p = strchr(p, ',')+1;
    buffer[0] = p[0];
    buffer[1] = p[1];
    buffer[2] = ':';
    buffer[3] = p[2];
    buffer[4] = p[3];
    buffer[5] = ':';
```

```
    buffer[6] = p[4];
    buffer[7] = p[5];
```

ignoramos milisegundos

```
    buffer[8] = ',';

    p = strchr(buffer+8, ',')+1;

    p = strchr(p, ',')+1;
    // encuentra latitud
    p = strchr(p, ',')+1;

    buffer[9] = '+';
    buffer[10] = p[0];
    buffer[11] = p[1];
    buffer[12] = ' ';
    strncpy(buffer+13, p+2, 7);
    buffer[20] = ',';

    p = strchr(buffer+21, ',')+1;
    if (p[0] == 'S')
      buffer[9] = '-';
```

encuentra longitud

```
    p = strchr(p, ',')+1;
    buffer[21] = '+';
    buffer[22] = p[0];
    buffer[23] = p[1];
    buffer[24] = p[2];
    buffer[25] = ' ';
    strncpy(buffer+26, p+3, 7);
    buffer[33] = ',';

    p = strchr(buffer+34, ',')+1;
    if (p[0] == 'W')
      buffer[21] = '-';
```

encuentra velocidad

```
    p = strchr(p, ',')+1;
    tmp = 0;
    if (p[0] != ',') {
```

convertimos la velocidad (viene en nudos)

```
      while (p[0] != '.' && p[0] != ',') {
        tmp *= 10;
        tmp += p[0] - '0';
        p++;
      }
      tmp *= 10;
      if (isdigit(p[1]))
        tmp += p[1] - '0';
      tmp *= 10;
      if (isdigit(p[2]))
      tmp += p[2] - '0';

      tmp *= 185; //la convertimos en km/h

    }
    tmp /= 100;

    buffer[34] = (tmp / 10000) + '0';
    tmp %= 10000;
    buffer[35] = (tmp / 1000) + '0';
    tmp %= 1000;
    buffer[36] = (tmp / 100) + '0';
    tmp %= 100;
    buffer[37] = '.';
    buffer[38] = (tmp / 10) + '0';
    tmp %= 10;
    buffer[39] = tmp + '0';

    buffer[40] = ',';
    p = strchr(p, ',')+1;
    // skip past bearing
    p = strchr(p, ',')+1;
```

mod para evitar problemas cuando falta algun dato (bill greiman)

```
    uint8_t date[6];
    for (uint8_t id = 0; id < 6; id++) date[id] = p[id];
```

formatea la fecha asi 2001-01-31

```
    buffer[41] = '2';
    buffer[42] = '0';
    buffer[43] = date[4];
    buffer[44] = date[5];
    buffer[45] = '-';
```

```
    buffer[46] = date[2];
    buffer[47] = date[3];
    buffer[48] = '-';
    buffer[49] = date[0];
    buffer[50] = date[1];
    buffer[51] = 0;

    if(f.write((uint8_t *) buffer, 51) != 51) {
      putstring_nl("no se ha podido escribir fix!");
            return;
    }
    Serial.print("Datos escritos en la SD: ");
    Serial.print(buffer);

    f.writeError = 0;
```

Aquí se añade la información de los sensores

```
    for (uint8_t ia = 0; ia < sensorCount; ia++) {
      Serial.print(',');   //escribimos por serie
      f.print(',');        //escribimos en el archivo
      uint16_t data = analogRead(ia);
      Serial.print(data);  //escribimos por serie
      f.print(data);       //escribimos en el archivo
    }
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

Imprimimos el archivo

```
    Serial.println();
    f.println();
    if (f.writeError || !f.sync()) {
      putstring_nl("no se ha podido escribir el dato!");
      error(4);
    }
    bufferidx = 0;
    return;
  }
  bufferidx++;
  if (bufferidx == BUFFSIZE-1) {
    Serial.print('!', BYTE);
    bufferidx = 0;
  }
}
```

```
}
```

Una vez que se cargue y el Arduino esté encendido se puede cambiar el interruptor del GPS a ON.

Obteniendo la posición, el led rojo se mantiene encendido y cada segundo empieza a guardar los datos en un archivo de datos separados por comas, encendiendo el led verde por cada escritura.

Si el led rojo parpadea es que el GPS no está arrojando datos válidos o también porque ocurre un error de escritura en la tarjeta. Se pueden ver los datos o errores conectando el Arduino al computador y se abre el monitor serial a 4800 bps.

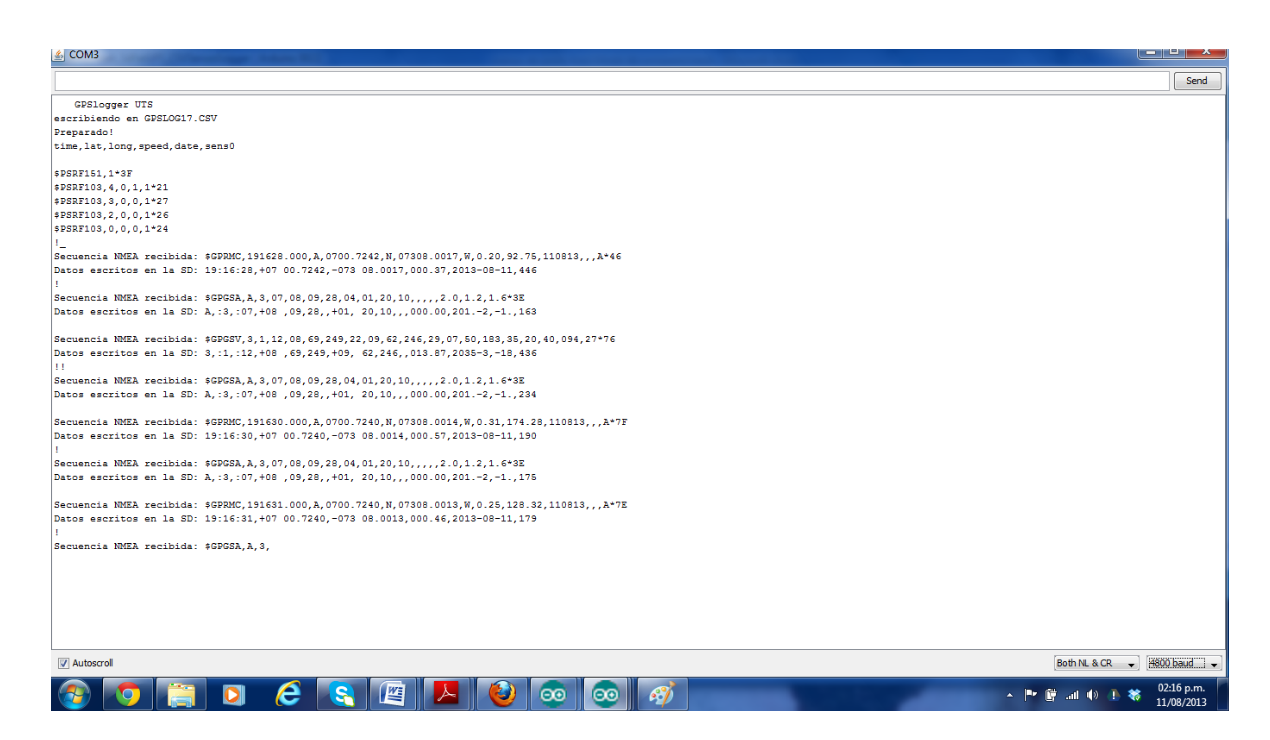


Figura 3.11 Datos NMEA
ANTOLINES, Jorge. "Datos NMEA". [Fotografía] (2013).

3.1.3 GPS EM406

El montaje del GPS es muy sencillo, lo que se requiere es soldar los headers a la tarjeta GPS, como se muestra en la siguiente figura:

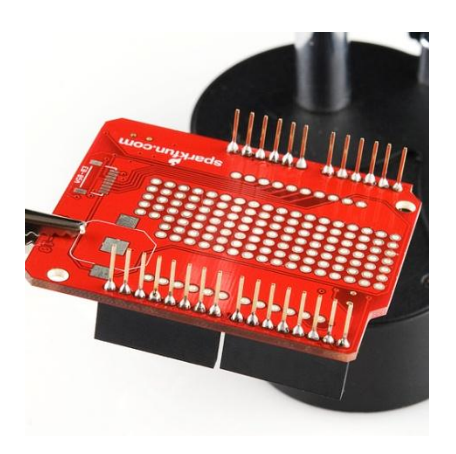


Figura 3.12 Soldadura de headers
MCI electronics. “Arduino Proto Shield kit”. [fotografía]. Tomado de:
http://www.olimex.cl/product_info.php?products_id=464

Para facilitar la ejecución del trabajo es necesario tener las siguientes herramientas:

Cautín o soldador
Soldadura
Crema para soldar

Para el montaje de todos los elementos, se requiere de:

- Una placa GPS
- 2x8 headers
- 2x6 headers

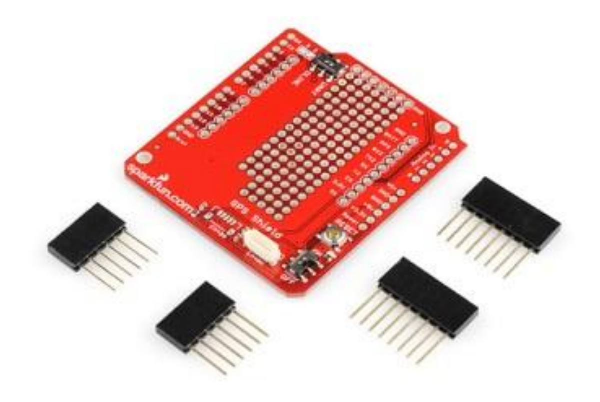


Figura 3.13 Componentes Shield GPS

MCI electronics. “Arduino Proto Shield kit”. [fotografía]. Tomado de: http://www.olimex.cl/product_info.php?products_id=464

SOLDADURA

Inicialmente se procede a soldar los encabezados, con el fin de que queden apilados de manera que se puedan insertar los puentes en las cabeceras para acceder a los pines no utilizados.

Se comienza por una soldadura de los seis conectores macho en uno de los seis orificios del GPS; se inserta uno de la cabecera de seis pines en la ubicación correcta.

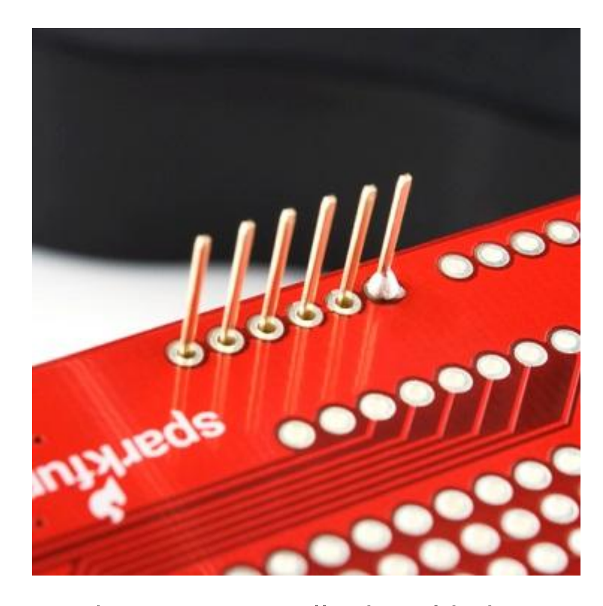


Figura 3.14 Detalle de soldadura

MCI electronics. “Arduino Proto Shield kit”. [fotografía]. Tomado de: http://www.olimex.cl/product_info.php?products_id=464

De esta manera, hay que asegurarse de que los pines de cabecera están saliendo de los orificios en ángulo recto, por lo que el GPS se conecta perfectamente en la placa principal del Arduino.

El motivo por el que se comienza con la soldadura de un solo pin es porque así se facilita la obtención de la alineación precisa y corregir a tiempo cualquier error o inconveniente.

En el momento en que se haya realizado la soldadura de un pin, se puede comprobar la alineación mediante la inserción de la pantalla del Arduino.

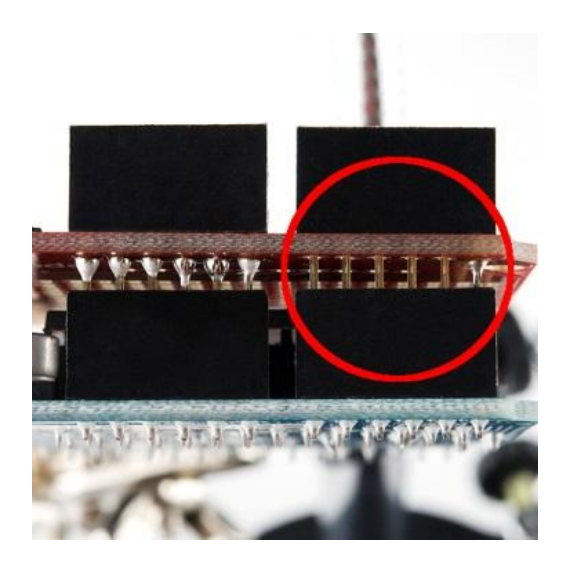


Figura 3.15 Alineación de pines
MCI electronics. “Arduino Proto Shield kit”. [fotografía]. Tomado de:
http://www.olimex.cl/product_info.php?products_id=464

Si la alineación del encabezado no está del todo bien, se debe calentar cuidadosamente la soldadura y mover la cabecera ligeramente. No se recomienda mover después de que se haya eliminado el calor, pues de esta manera se puede averiar o romper la cabecera.

Al alinear la cabecera, se procede a soldar otro pin, lo recomendable es soldar uno del otro extremo de la cabecera, ya que así éstos quedarán alineados, fijos y no van a cambiar.

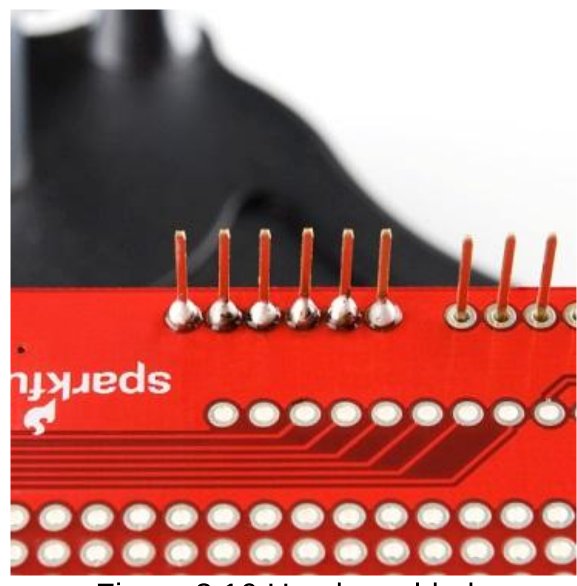


Figura 3.16 Header soldado
MCI electronics. "Arduino Proto Shield kit". [fotografía]. Tomado de: http://www.olimex.cl/product_info.php?products_id=464

Finalmente, se repite el proceso anterior para los 3 ejes restantes. Se deben soldar los encabezados de 6 pines en las ranuras de 6 pines y los encabezados de 8 pines en las ranuras de 8 pines en los bordes del Arduino.

Una vez se haya finalizado la soldadura, queda un resultado como el siguiente:



Figura 3.17 Trabajo final

MCI electronics. “Arduino Proto Shield kit”. [fotografía]. Tomado de:
http://www.olimex.cl/product_info.php?products_id=464

Cuando la placa está completa, simplemente se enchufa un módulo GPS EM-406 al conector, se puede fijar el GPS a la placa con un poco de cinta adhesiva doble faz.



Figura 3.18 GPS fijado a la placa
MCI electronics. “Arduino Proto Shield kit”. [fotografía]. Tomado de:
http://www.olimex.cl/product_info.php?products_id=464

Por último se conecta la placa en el Arduino y está preparado para trabajar.



Figura 3.19 Arduino Uno + Shield GPS
MCI electronics. "Arduino Proto Shield kit". [fotografía]. Tomado de:
http://www.olimex.cl/product_info.php?products_id=464

3.2 SOFTWARE

Se comienza por entender los conceptos básicos y comandos AT con los que trabaja Arduino.

3.2.1 TIPOS DE DATOS

Los tipos de datos que se trabajan son iguales a los de C++, con una pequeña diferencia en la escritura del contenido de las bibliotecas, debido a que Arduino Uno, actualiza constantemente, sus versiones de IDE y eso hace que una palabra como byte funcione en ciertas versiones.

TIPOS DE DATOS	DESCRIPCIÓN
Byte	almacena un valor numérico de 8 bits sin puntos decimales. Tienen un rango de 0 a 255.
Char	es un tipo de dato que ocupa un byte de memoria y almacena un valor de carácter. Los caracteres literales se escriben con comillas simples: 'A' (para varios caracteres -strings- se utiliza

	dobles comillas "ABC"). De todas maneras los caracteres son almacenados como números. Se puede ver su codificado en la tabla ASCII. Con esto se puede entender que es posible realizar cálculos aritméticos con los caracteres, en este caso se utiliza el valor ASCII del carácter (por ejemplo 'A' + 1 tiene el valor de 66, ya que el valor ASCII de la letra mayúscula A es 65) El tipo de datos char tiene signo, esto significa que codifica números desde -128 hasta 127. Para un dato de un byte (8 bits), utiliza el tipo de dato "byte".
Int	enteros son los tipos de datos primarios para almacenamiento de números sin puntos decimales y almacenan un valor de 16 bits con un rango de -32,768 a 32,767.
Long	tipo de datos de tamaño extendido para enteros largos, sin puntos decimales, almacenados en un valor de 32 bits con un rango de -2,146,483,648 a 2,147,483,647.
Float	un tipo de datos para números en punto flotante, o números que tienen un punto decimal. Los números en punto flotante tienen mayor resolución que los enteros y se almacenan como valor de 32 bits con un rango de -3.4028235E+38 a 3.4028235E+38.

Tabla 3.1. Tipos de datos

Constantes

El lenguaje Arduino tiene unos cuantos valores predefinidos que se llaman constantes. Se usan para hacer los programas más legibles. Las constantes se clasifican en grupos.

- TRUE / FALSE
- HIGH / LOW
- INPUT / OUTPUT

CONSTANTES	DESCRIPCIÓN
True/False	estas son constantes booleanas que definen niveles lógicos. FALSE se define como 0 (cero) mientras TRUE es 1 o un valor distinto de 0.
High/low	estas constantes definen los niveles de pin como HIGH o LOW y se usan cuando se leen o se escriben los pines digitales. HIGH está definido como el nivel 1 lógico, ON ó 5 V, mientras que LOW es el nivel lógico 0, OFF ó 0 V.
Input/output	constantes usadas con la función pinMode() para definir el modo de un pin digital como INPUT u OUTPUT.
Conversión de datos	permite cambiar un valor dado a otro de un tipo de dato específico. char(x), byte(x), int(x), long(x), float(x)

Tabla 3.2. Constantes

Tabla de Ciclos

CICLOS	**DESCRIPCIÓN**
If	comprueban si cierta condición ha sido alcanzada y ejecutan todas las sentencias dentro de las llaves si la declaración es cierta. Si es falsa el programa ignora la sentencia.
For	se usa para repetir un bloque de declaraciones encerradas en llaves un número específico de veces. Un contador de incremento se usa a menudo para incrementar y terminar el bucle. Hay tres partes separadas por punto y coma (;), en la cabecera del

	bucle. La inicialización de una variable local, o contador de incremento, sucede primero y una sola una vez. Cada vez que pasa el bucle, la condición siguiente es comprobada. Si la condición devuelve TRUE, las declaraciones y expresiones que siguen se ejecutan y la condición se comprueba de nuevo. Cuando la condición se vuelve FALSE, el bucle termina.
While	se repetirá continuamente, e infinitamente, hasta que la expresión dentro del paréntesis se vuelva falsa. Algo debe cambiar la variable comprobada o el bucle while nunca saldrá. Lo que modifique la variable puede estar en el código, como una variable que se incrementa, o ser una condición externa, como el valor que da un sensor.
Do…While	es un bucle que trabaja de la misma forma que el bucle while, con la excepción de que la condición es testeada al final del bucle, por lo que el bucle do. . . while siempre se ejecutará al menos una vez.
Break	es usado para salir de los bucles **do**, **for**, o **while**, pasando por alto la condición normal del bucle.

Tabla 3.3. Ciclos

3.2.2 IDE ARDUINO

Se debe trabajar con la versión de IDE Arduino 0022, debido a que las actualizaciones del IDE se realizan constantemente por los creadores y esto implica cambios en algunos comandos.

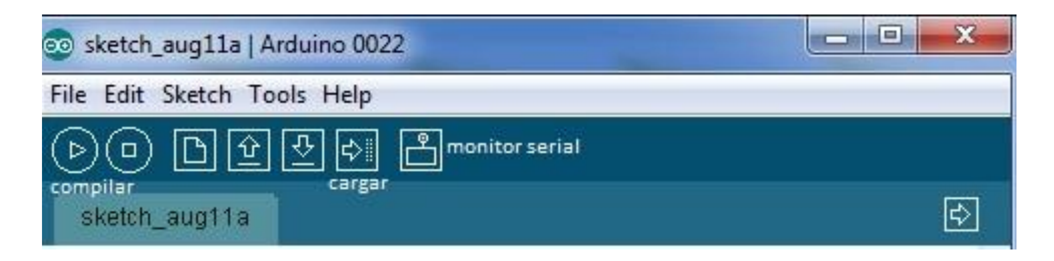


Figura 3.20 IDE de Arduino

ANTOLINES, Jorge. "IDE de Arduino". [Fotografía]. (2013)

Después de cargar cualquier código, se compila el código hasta que aparezca la siguiente respuesta en el IDE:

```
Done uploading.
Binary sketch size: 13752 bytes (of a 32256 byte maximum)
384
```

Figura 3.21 Carga exitosa

ANTOLINES, Jorge. "Carga exitosa IDE-Arduino". [Fotografía] (2013)

Esto significa que en el código no hay ningún error de escritura y se puede cargar en la tarjeta Arduino.

El interruptor de la placa GPS debe estar en ON y el interruptor conmutable debe estar en UART.

Figura 3.22 Ejemplo de carga en el IDE

ANTOLINES, Jorge. “Ejemplo carga en el IDE”. [Fotografía]. (2013)

Cuando el IDE termina de cargar el código en el Arduino el led conectado al pin 13 y GND parpadea, indicando que la carga ha sido exitosa.

3.2.3 APP (Ver anexo – Guía 0)

- Se Crea el archivo PHP llamado gpskml, que se encarga de re-direccionar nuestro aplicativo a gpsvisualizer.



Figura 3.23 APP

- Se sube el archivo a un servidor en internet (su uso el comyr.com/ - ver tutorial en el siguiente link http://www.youtube.com/watch?v=vzUz6ElxHY4), en el que se crea un usuario y dominio para cargar toda la información.

- Se crea el aplicativo para el dispositivo móvil, en donde se hace uso de la página http://www.appsgeyser.com/ (ver tutorial en http://www.youtube.com/watch?v=6L8YejfFOeQ)..

3.3 PRUEBAS

3.3.1 PRUEBA 1

Una de las pruebas realizadas arroja los siguientes resultados:

Estado del cielo: Parcialmente nublado

Vehiculo: Motocicleta

RUTA:

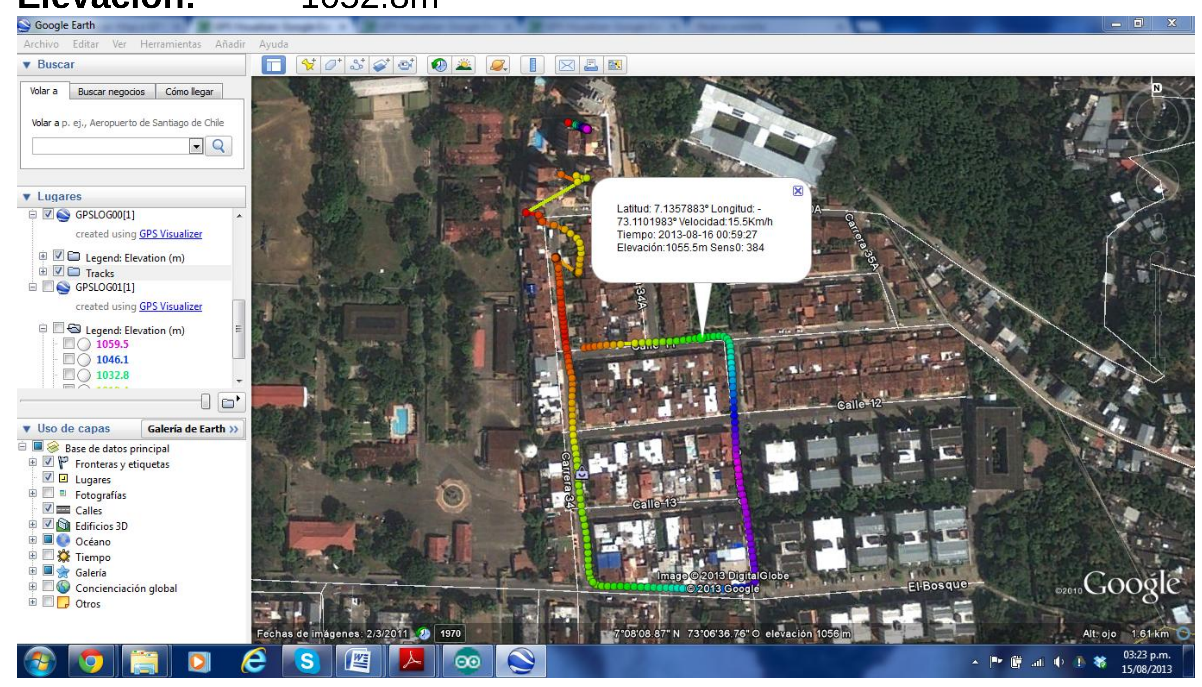
Inicio: Desde la calle 10 N° 34 -15 TORRES DE LOS PINOS

Fin: Calle 11 N° 34 – 28

Se observa una alta precisión en la obtención de datos reflejados en Google Earth.

Datos obtenidos:

Latitud: 7.1365767º
Longitud: -73.11119º
Velocidad: 13.8Km/h
Tiempo: 2013-08-16
Elevación: 1052.8m



Archivo de PRUEBA 1 en anexos, nombre: GPSLOG00

3.3.2 PRUEBA 2:

Estado del cielo: Parcialmente nublado

Vehiculo: Motocicleta

RUTA:

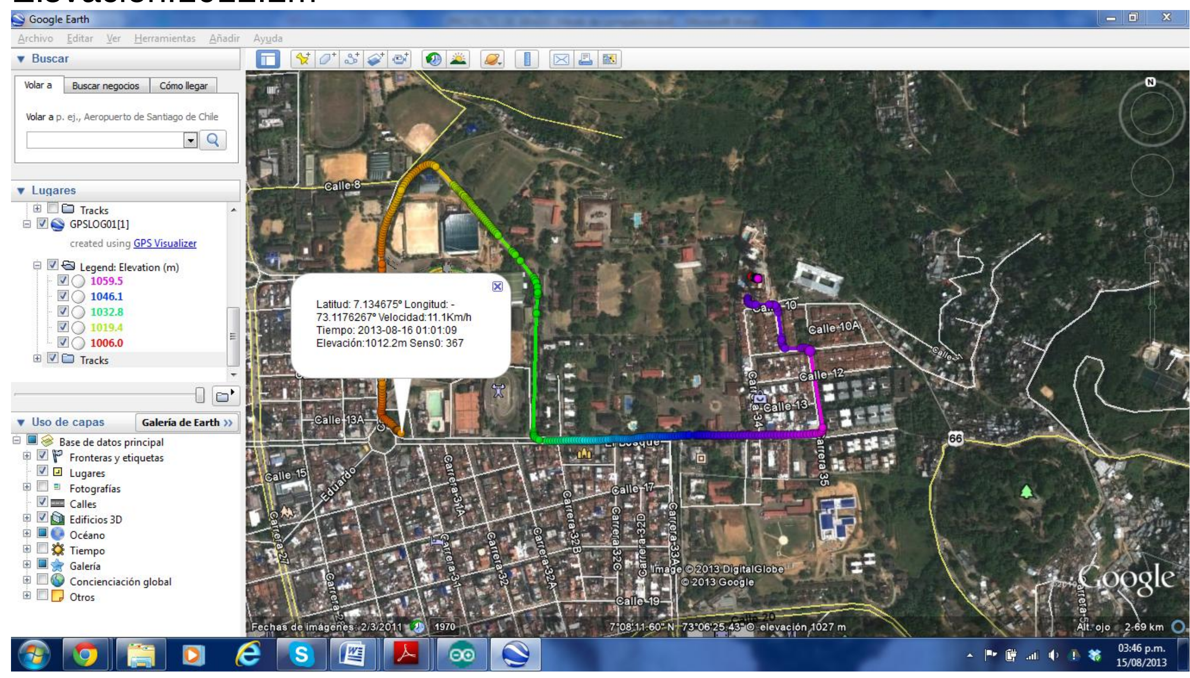
Inicio: Glorieta del Estado Alfonso López

Fin: calle 10 N° 34 -15 TORRES DE LOS PINOS

Se observa una alta precisión en la obtención de datos reflejados en Google Earth.

Datos obtenidos:

Latitud: 7.134675º
Longitud: -73.1176267º
Velocidad:11.1Km/h
Tiempo: 2013-08-16
Elevación:1012.2m



Archivo de PRUEBA 2 en anexos, nombre: GPSLOG01

CAPÍTULO 4

4.1 CONCLUSIONES

- Arduino es una plataforma flexible escalable y adaptable, que permite crear interfaces para desarrollar múltiples aplicaciones, dirigidas a robótica, telecomunicaciones, automatización y casi cualquier labor, que tenga que ver con proyección tecnológica, aplicada al aprendizaje.
- El Arduino Uno, tiene una mayor velocidad de recepción con el microcontrolador ATmega328, comparado con cualquier PIC.
- Arduino Uno, facilita la carga y la modificación del código, sin necesidad de un quemador de microcontroladores.
- Arduino Uno, ensambla perfectamente con módulos adicionales (Shields).
- El módulo GPS EM406, es preciso, debido a que arroja resultados aunque el cielo esté parcialmente nublado; es recomendable para vehículos terrestres ya que su frecuencia es de 1Hz, mientras que para aeomodelismo no es óptimo, pues para éste se requiere una frecuencia de 5Hz como mínimo.
- El diseño de la APP permite visualizar los datos obtenidos del GPS en cualquier dispositivo móvil con sistema operativo Android y la interfaz en segundo plano utilizado es Google Earth, con el fin de utilizar los mapas cargados en el servidor y evitar que ocupe espacio en la memoria del dispositivo móvil cargando los mapas independientemente.

BIBLIOGRAFÍA

- BANZI, Massimo. Introducción a Arduino. Editorial: Anaya Multimedia (2012)
- SMICHMIDT, Maik. Arduino a Quick-Start Guide. Editorial: The pragmatic Bookshelf (2011). Primera Edición
- CEBALLOS, Francisco Javier. Programación orientada a objetos con C++. Editorial: Ra-ma (2007)
- ARNALICH, Santigo, URRUELA, Julio. Arnalich. Editorial: Arnalich. Water and habitat (2012). Primera Edición
- CUELLO, Javier, VITTONE, José. Diseñando APPS para móviles. Editorial: José Vittone ISBN: 8461649338.

WEB-BIBLIOGRAFÍA

- Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, and David Mellis, “Arduino”: (http://www.arduino.cc/es/)
- ¿Qué es Arduino?: (http://arduino.cc/es/Guide/Introduction)
- Df Robot Drive The Future: (http://www.dfrobot.com/index.php?route=product/product&product_id=673#.UFkBqo2TuN4)
- Tecnoproject Seguridad Electrónica e Informática:
- rcravens “Create a GPS Data Logger Using The Arduino”: (http://blog.bobcravens.com/2010/08/create-a-gps-data-logger-using-the-arduino/)
- David Cuartielles “Proyectos Arduino”: (http://proyectoarduino.wordpress.com/%C2%BFque-es-arduino/)
- David Cuartielles “¿Qué es Arduino?”, (http://proyectoarduino.wordpress.com/%C2%BFque-es-arduino/)
- Arduino. “Arduino Uno”: http://arduino.cc/en/Main/arduinoBoardUno

RECOMENDACIONES

Documentarse antes de comprar cualquier módulo.

Si se van a realizar envíos de datos en tiempo real mediante un GPRS es necesario entender cómo se crea un servidor PHP, una tabla MYSQL, y algún código que haga la interconexión entre estos dos, como por ejemplo, PYTHON, con el fin de elegir el módulo GPRS adecuado, aquí un link para el desarrollo de un prototipo en el futuro.

http://www.cooking-hacks.com/index.php/documentation/tutorials/geolocation-tracker-gprs-gps-geoposition-sim908-arduino-raspberry-pi?utm_source=banner_sim908&utm_medium=banner

ÍNDICE DE TABLAS